

Coding Methods for Emerging Storage Systems

Lara Dolecek and Anxiao (Andrew) Jiang

Department of Electrical Engineering, UCLA,
Department of Computer Science, Texas A&M



Asilomar Conference Tutorial, Nov. 2012

Data storage technologies

Storage technologies have revolutionized the way we create, manipulate, and use data.

- 1950-1960s: Punch cards and magnetic tape
- 1970-1980s: Optical recording (DVDs), floppy disks
- 1990-2000s: Hard disk drives
- 2010+: Era of Flash drives



Non-volatile memories (NMVs) are increasingly ubiquitous

Smart Phones



Tablets and E-Readers

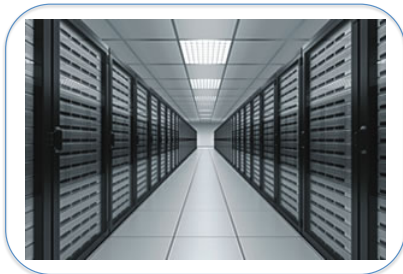


Non-volatile memories (NMVs) are increasingly ubiquitous

Smart Phones



Tablets and E-Readers



Data Centers

Non-volatile memories (NMVs) are increasingly ubiquitous

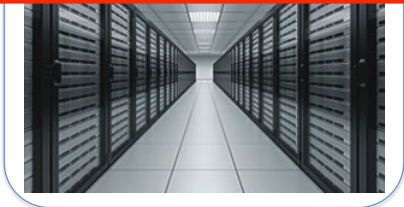
Smart Phones



Tablets and E-Readers



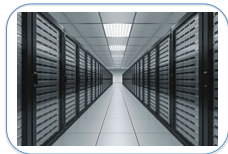
With the on-going data revolution, applications from consumer electronics to data centers abound.



Data Centers

NVMs today

- NVMs are solid state memory devices that need not be periodically refreshed.
- SSDs have 20x faster read than HDDs due to negligible seek time and consume up to 90% less power.
- Market size over 70 billion USD worldwide.
- Key NVM technologies
 - Flash Memories
 - Phase Change Memories
 - STTRAMs, Memristors



NVMs of the future

Data revolution demands NVMs with

- Increased density at reduced size,
- Improved reliability at increased noise levels,
- Improved endurance and retention,
- Faster read/write access time,
- Compact cross-layer architectures.

NVMs of the future

Data revolution demands NVMs with

- Increased density at reduced size,
- Improved reliability at increased noise levels,
- Improved endurance and retention,
- Faster read/write access time,
- Compact cross-layer architectures.

Novel coding methodologies hold promise to address all these challenges.

What the Future May Bring:



What is this tutorial about ?

Today we will learn about

- Physical characteristics of emerging non-volatile memories,

What is this tutorial about ?

Today we will learn about

- Physical characteristics of emerging non-volatile memories,
- Channel models associated with these technologies,

What is this tutorial about ?

Today we will learn about

- Physical characteristics of emerging non-volatile memories,
- Channel models associated with these technologies,
- Various recent developments in coding for memories, and

What is this tutorial about ?

Today we will learn about

- Physical characteristics of emerging non-volatile memories,
- Channel models associated with these technologies,
- Various recent developments in coding for memories, and
- Open problems and future directions in communication techniques for memories.

- 1 Channel Models
- 2 Error Correcting Codes
- 3 Codes for Rewriting Data
- 4 Rank Modulation
- 5 Constrained Coding
- 6 Summary and Future Directions

Outline

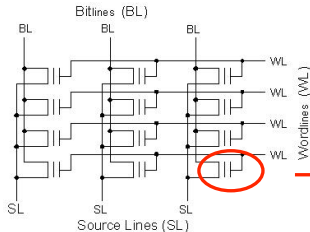
Channel Models
Error Correcting Codes
Codes for Rewriting Data
Rank Modulation
Constrained Coding
Summary and Future Directions

Part I

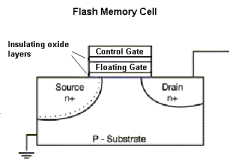
Flash technology

- A flash memory block is an array of $2^{20} \sim$ million cells.
- A cell is a floating-gate transistor

Cell array in a flash memory



A flash memory cell (Floating gate)



Flash technology

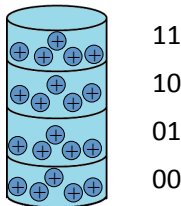
Terminology

- Single level cell (SLC) stores 1 bit per cell (2 levels)
- Multiple level cell (MLC) stores 2 bit per cell (4 levels)
- Triple level cell (TLC) stores 3 bit per cell (8 levels)
- Cell levels correspond to the voltage induced by the number of electrons stored on the gate.

Flash technology

Terminology

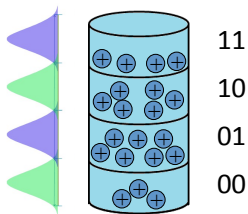
- Single level cell (SLC) stores 1 bit per cell (2 levels)
- Multiple level cell (MLC) stores 2 bit per cell (4 levels)
- Triple level cell (TLC) stores 3 bit per cell (8 levels)
- Cell levels correspond to the voltage induced by the number of electrons stored on the gate.



Flash technology

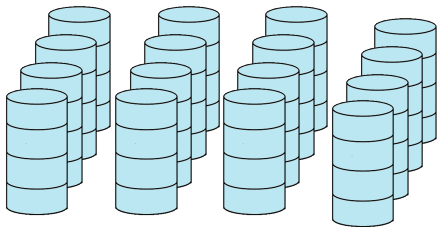
Terminology

- Single level cell (SLC) stores 1 bit per cell (2 levels)
- Multiple level cell (MLC) stores 2 bit per cell (4 levels)
- Triple level cell (TLC) stores 3 bit per cell (8 levels)
- Cell levels correspond to the voltage induced by the number of electrons stored on the gate.



Flash programming – example 1

Flash block

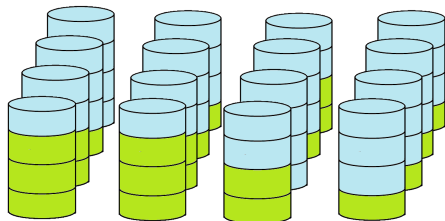


Initial state

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Flash programming – example 1

Flash block

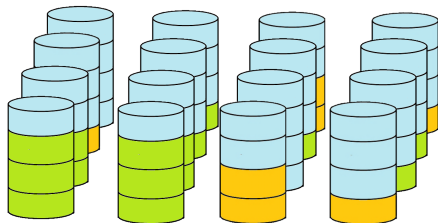


Final state

0	1	2	1
1	1	1	1
2	2	0	1
3	3	2	1

Flash programming – example 2

Flash block

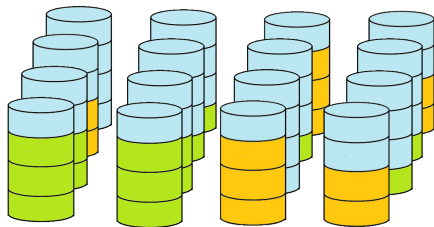


Initial state

0	1	2	1
1	1	1	1
2	2	0	1
3	3	2	1

Flash programming – example 2

Flash block

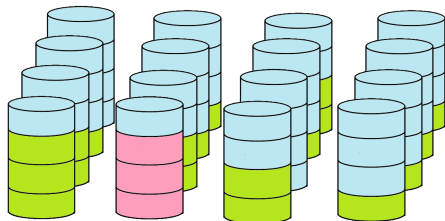


Final state

0	1	3	2
2	1	1	1
2	2	0	1
3	3	3	2

Flash programming – example 3

Flash block

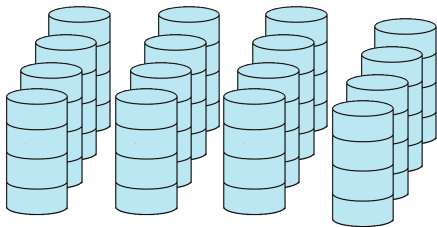


Initial state

0	1	2	1
1	1	1	1
2	2	0	1
3	3	2	1

Flash programming – example 3

Flash block

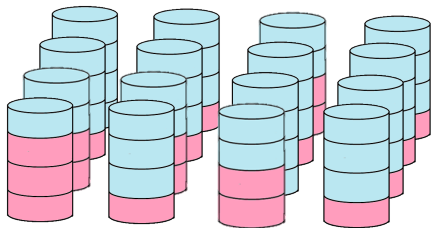


Intermediate state

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Flash programming – example 3

Flash block



Final state

0	1	2	1
1	1	1	1
2	2	0	1
3	1	2	1

Flash endurance

- Frequent block erases cause so-called “wearout” due to charge loss on the gates.
- Flash memory lifetime is commonly expressed in terms of the number of program and erase (P/E) cycles allowed before memory is deemed unusable.
- Lifetime:
 - SLC: $\approx 10^6$ P/E cycles
 - MLC: $\approx 10^5$ P/E cycles
 - TLC: $\approx 10^4$ P/E cycles
- With frequent writes 2GB TLC lasts less than 3 months!

Flash inter-cell interference

Floating gate to floating gate inter-cell coupling

- Change in charge in one cell affects voltage threshold of a neighboring cell.
- During write/read, stressed (victim) cell appears weakly programmed.

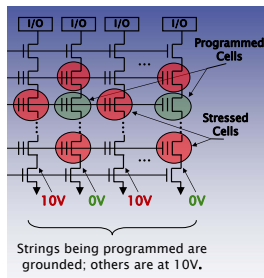
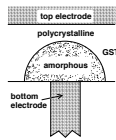


Figure: Courtesy Micron, Flash Memory Summit Presentation, 2007.

Phase change memories (PCM) model

Two states

- Amorphous – high resistance
- Polycrystalline – low resistance



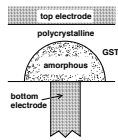
Phase change memories (PCM) model

Pros

- Erases on the cell level
- Faster access
- More P/E cycles, longer lifetime

Two states

- Amorphous – high resistance
- Polycrystalline – low resistance



Phase change memories (PCM) model

Pros

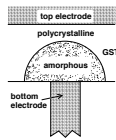
- Erases on the cell level
- Faster access
- More P/E cycles, longer lifetime

Cons

- Not nearly as dense
- Resistance and voltage drift
- Higher processing cost

Two states

- Amorphous – high resistance
- Polycrystalline – low resistance



Issues



- Delay onset of errors
- Improve reliability
- Improve write access
- Increase storage capacity
- Reduce interference

Techniques

- Error correction codes
- Codes for rewriting data
- Rank modulation
- Constrained coding

Further reading (1/2)

Modeling of Flash



-  L. M. Grupp *et al.* “Beyond the datasheet: using test beds to probe non-volatile memories’ dark secrets,” IEEE Globecom, 2010.
-  L. M. Grupp *et al.* “Characterizing flash memory: anomalies, observations, and applications,” IEEE/ACM MICRO, 2009.

Inter-cell Interference

-  J-D. Lee, S-H Hur, and J-D. Choi, “Effects of floating-gate interference on NAND flash memory cell operation”, IEEE Electron Device Letters, 2002.

Further reading (2/2)

PCM Model

-  M. Franceschini *et al.*, “A Communication-theoretic approach to phase change storage,” IEEE ICC, 2010.
-  L. Lastras-Montañaño *et al.*, “On the lifetime of multilevel memories,” IEEE ISIT, 2009.

Cross-Technology Comparisons

-  G. W. Burr *et al.*, “Overview of candidate device technologies for storage-class memory,” IBM J. on R&D, 2008.

Resources at the Annual Flash Memory Summit Page
<http://www.flashmemorysummit.com/>

Error Correcting Codes

Issues

- Delay onset of errors
- Improve reliability
- Improve write access
- Increase storage capacity
- Reduce interference

Techniques

- Error correction codes
- Codes for rewriting data
- Rank modulation
- Constrained coding

Issues

- Delay onset of errors
- Improve reliability
- Improve write access
- Increase storage capacity
- Reduce interference

Techniques

- **Error correction codes**
- Codes for rewriting data
- Rank modulation
- Constrained coding

Issues

- Delay onset of errors
- Improve reliability
- Improve write access
- Increase storage capacity
- Reduce interference

Techniques

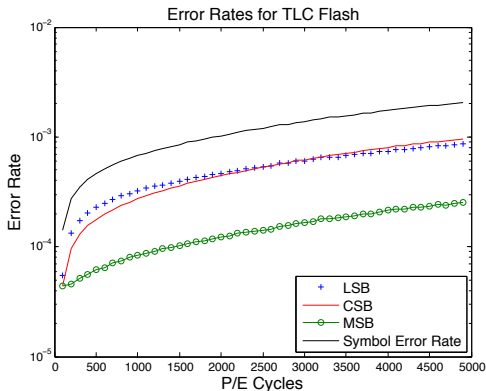
- Error correction codes
- Codes for rewriting data
- Rank modulation
- Constrained coding

Error correction is a must for memories

Conventional error correcting codes are in widespread use

- Early SLC technologies used basic Hamming codes
 - Simple to implement but offer limited protection
- BCH codes have become increasingly popular
 - Excellent algebraic codes but increasingly inadequate for NVM channels
- LDPC codes are now being actively investigated
 - Excellent graph-based codes but no performance guarantees in general

Raw error rate for TLC flash



Data collected in Swanson Lab, UCSD.

LSB: least significant bit
 CSB: center significant bit
 MSB: most significant bit

Table: Mapping between Voltage Levels and Triple-bit Words

Voltage Level	Triple-bit Word
0	111
1	110
2	100
3	101
4	001
5	000
6	010
7	011

Error patterns within a TLC cell

Number of bits in symbol that err	Percentage of errors
1	0.9617
2	0.0314
3	0.0069

- Standard error-correction codes are designed to correct all symbol errors.

Error patterns within a TLC cell

Number of bits in symbol that err	Percentage of errors
1	0.9617
2	0.0314
3	0.0069

- Standard error-correction codes are designed to correct all symbol errors.
- Usage of standard codes: overkill in terms of redundancy.

Error patterns within a TLC cell

Number of bits in symbol that err	Percentage of errors
1	0.9617
2	0.0314
3	0.0069

- Standard error-correction codes are designed to correct all symbol errors.
- Usage of standard codes: overkill in terms of redundancy.
- Instead, design codes for the observed intracell variability.

Channel coding preliminaries

- A code \mathcal{C} is defined over the alphabet \mathcal{A} of size $q = 2^m$, where m is a positive integer and q is the number of levels in a flash cell.

Channel coding preliminaries

- A code \mathcal{C} is defined over the alphabet \mathcal{A} of size $q = 2^m$, where m is a positive integer and q is the number of levels in a flash cell.
- A Flash cell value is represented by a symbol in $GF(2)^m$.

Channel coding preliminaries

- A code \mathcal{C} is defined over the alphabet \mathcal{A} of size $q = 2^m$, where m is a positive integer and q is the number of levels in a flash cell.
- A Flash cell value is represented by a symbol in $GF(2)^m$.
- A block of n flash cells is represented by a codeword c in \mathcal{C} , taking value in $(GF(2)^m)^n$.

Channel coding preliminaries

- A code \mathcal{C} is defined over the alphabet \mathcal{A} of size $q = 2^m$, where m is a positive integer and q is the number of levels in a flash cell.
- A Flash cell value is represented by a symbol in $GF(2)^m$.
- A block of n flash cells is represented by a codeword c in \mathcal{C} , taking value in $(GF(2)^m)^n$.
- An example:

$$m = 3, \mathcal{A} = GF(8), n = 5$$

$$(45702) \rightarrow (100 \ 101 \ 111 \ 000 \ 010)$$

Flash block \rightarrow codeword

Bit-error patterns

Definition (Bit-Error Vector)

Given positive integers t and ℓ , the vector $\mathbf{e} = (e_1, \dots, e_n) \in (GF(2)^m)^n$ is a $[t; \ell]_{2^m}$ -bit-error vector when

Bit-error patterns

Definition (Bit-Error Vector)

Given positive integers t and ℓ , the vector $\mathbf{e} = (e_1, \dots, e_n) \in (GF(2)^m)^n$ is a $[t; \ell]_{2^m}$ -bit-error vector when

- 1 The number of non-zero symbols e_i 's, $1 \leq i \leq n$, is at most t .

Bit-error patterns

Definition (Bit-Error Vector)

Given positive integers t and ℓ , the vector $\mathbf{e} = (e_1, \dots, e_n) \in (GF(2)^m)^n$ is a $[t; \ell]_{2^m}$ -bit-error vector when

- 1 The number of non-zero symbols e_i 's, $1 \leq i \leq n$, is at most t .
- 2 For each symbol e_i , $1 \leq i \leq n$, there are at most ℓ non-zero bits.

Bit-error example

- Suppose the vector \mathbf{x} of length 6 with 3-bit symbols was transmitted (stored):

$$\mathbf{x} = (000 \ 110 \ 010 \ 101 \ 000 \ 111)$$

Bit-error example

- Suppose the vector \mathbf{x} of length 6 with 3-bit symbols was transmitted (stored):

$$\mathbf{x} = (000 \ 110 \ 010 \ 101 \ 000 \ 111)$$

- Suppose the vector \mathbf{y} was received (retrieved):

$$\mathbf{y} = (101 \ 110 \ 000 \ 101 \ 000 \ 011).$$

Bit-error example

- Suppose the vector \mathbf{x} of length 6 with 3-bit symbols was transmitted (stored):
$$\mathbf{x} = (000\ 110\ 010\ 101\ 000\ 111)$$
- Suppose the vector \mathbf{y} was received (retrieved):
$$\mathbf{y} = (101\ 110\ 000\ 101\ 000\ 011).$$
- Then $[3; 2]_{2^3}$ -bit-errors are said to have occurred since there are **3 symbols in error** and for each symbol **at most 2 bits** are in error.

Graded bit-error patterns

Definition (Graded Bit-Error Vector)

Given positive integers t_1 , t_2 , l_1 and l_2 , $l_1 < l_2$, the vector $\mathbf{e} = (e_1, \dots, e_n) \in (GF(2)^m)^n$ is a $[t_1, t_2; l_1, l_2]_{2^m}$ -graded bit-error vector when

Graded bit-error patterns

Definition (Graded Bit-Error Vector)

Given positive integers t_1 , t_2 , l_1 and l_2 , $l_1 < l_2$, the vector $\mathbf{e} = (e_1, \dots, e_n) \in (GF(2)^m)^n$ is a $[t_1, t_2; l_1, l_2]_{2^m}$ -graded bit-error vector when

- 1 The number of non-zero symbols e_i 's, $1 \leq i \leq n$, is at most $t_1 + t_2$.

Graded bit-error patterns

Definition (Graded Bit-Error Vector)

Given positive integers t_1 , t_2 , l_1 and l_2 , $l_1 < l_2$, the vector $\mathbf{e} = (e_1, \dots, e_n) \in (GF(2)^m)^n$ is a $[t_1, t_2; l_1, l_2]_{2^m}$ -graded bit-error vector when

- 1 The number of non-zero symbols e_i 's, $1 \leq i \leq n$, is at most $t_1 + t_2$.
- 2 For each symbol e_i , there are at most l_2 non-zero bits.

Graded bit-error patterns

Definition (Graded Bit-Error Vector)

Given positive integers t_1 , t_2 , l_1 and l_2 , $l_1 < l_2$, the vector $\mathbf{e} = (e_1, \dots, e_n) \in (GF(2)^m)^n$ is a $[t_1, t_2; l_1, l_2]_{2^m}$ -graded bit-error vector when

- 1 The number of non-zero symbols e_i 's, $1 \leq i \leq n$, is at most $t_1 + t_2$.
- 2 For each symbol e_i , there are at most l_2 non-zero bits.
- 3 There are at most t_2 symbols that have more than l_1 non-zero bits each.

Graded bit-error example

- Suppose the vector \mathbf{x} of length 6 with 3-bit symbols was transmitted (stored):

$$\mathbf{x} = (000 \ 110 \ 010 \ 101 \ 000 \ 111)$$

Graded bit-error example

- Suppose the vector \mathbf{x} of length 6 with 3-bit symbols was transmitted (stored):

$$\mathbf{x} = (000 \ 110 \ 010 \ 101 \ 000 \ 111)$$

- Suppose the vector \mathbf{y} was received (retrieved):

$$\mathbf{y} = (101 \ 110 \ 000 \ 101 \ 000 \ 011).$$

Graded bit-error example

- Suppose the vector \mathbf{x} of length 6 with 3-bit symbols was transmitted (stored):

$$\mathbf{x} = (000 \ 110 \ 010 \ 101 \ 000 \ 111)$$

- Suppose the vector \mathbf{y} was received (retrieved):

$$\mathbf{y} = (101 \ 110 \ 000 \ 101 \ 000 \ 011).$$

- Then $[2, 1; 1, 2]_{2^3}$ -graded-bit-errors are said to have occurred since there are $2 + 1$ **symbols in error** where there are **at most 2 bits in error** for each symbol and there is only **1 symbol that has more than 1 bit** in error.

Error-correcting codes

Definition (Bit-Error-Correcting Code)

A 2^m -ary linear code \mathcal{C} is a $[t; \ell]_{2^m}$ -**bit-error-correcting code** if it can correct every $[t; \ell]_{2^m}$ -bit error vector.

Error-correcting codes

Definition (Bit-Error-Correcting Code)

A 2^m -ary linear code \mathcal{C} is a $[t; \ell]_{2^m}$ -**bit-error-correcting code** if it can correct every $[t; \ell]_{2^m}$ -bit error vector.

Definition (Graded Bit-Error-Correcting Code)

A 2^m -ary code \mathcal{C} is a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$ -**graded-bit-error-correcting code** if it can correct every $[t_1, t_2; \ell_1, \ell_2]_{2^m}$ -graded bit error vector.

Recall: tensor product

Definition (Tensor Product)

Let $A \in GF(q)^{m \times n}$, $B \in GF(q)^{p \times r}$. Then the *tensor product* of A and B is defined as the matrix

$$A \otimes B = \begin{pmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{pmatrix} \in GF(q)^{mp \times nr}$$

Recall: tensor product

Definition (Tensor Product)

Let $A \in GF(q)^{m \times n}$, $B \in GF(q)^{p \times r}$. Then the *tensor product* of A and B is defined as the matrix

$$A \otimes B = \begin{pmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{pmatrix} \in GF(q)^{mp \times nr}$$

Example:

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 \end{pmatrix}, B = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

Recall: tensor product

Definition (Tensor Product)

Let $A \in GF(q)^{m \times n}$, $B \in GF(q)^{p \times r}$. Then the *tensor product* of A and B is defined as the matrix

$$A \otimes B = \begin{pmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{pmatrix} \in GF(q)^{mp \times nr}$$

Example:

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 \end{pmatrix}, B = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

$$A \otimes B = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}$$

Tensor product codes [1]

Theorem (Construction 1)

- Let H_1 be the parity check matrix of a $[m, k_1, \ell]_2$ code C_1 (standard $[n, k, e]$ notation).

Tensor product codes [1]

Theorem (Construction 1)

- Let H_1 be the parity check matrix of a $[m, k_1, \ell]_2$ code \mathcal{C}_1 (standard $[n, k, e]$ notation).
- Let H_2 be the parity check matrix of a $[n, k_2, t]_{2^{m-k_1}}$ code \mathcal{C}_2 defined over the alphabet of size $GF(2)^{m-k_1}$.

Tensor product codes [1]

Theorem (Construction 1)

- Let H_1 be the parity check matrix of a $[m, k_1, \ell]_2$ code \mathcal{C}_1 (standard $[n, k, e]$ notation).
- Let H_2 be the parity check matrix of a $[n, k_2, t]_{2^{m-k_1}}$ code \mathcal{C}_2 defined over the alphabet of size $GF(2)^{m-k_1}$.
- Then, $H_A = H_2 \otimes H_1$ is the parity check matrix of a $[t; \ell]_{2^m}$ -bit-error-correcting code \mathcal{C}_A of length n .

Example of a tensor product code

- Consider the parity check matrix H_1 of a $[3, 1, 1]_2$ code \mathcal{C}_1 :

$$H_1 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

Example of a tensor product code

- Consider the parity check matrix H_1 of a $[3, 1, 1]_2$ code \mathcal{C}_1 :

$$H_1 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

- Consider the parity check matrix H_2 of a $[4, 2, 1]_4$ code \mathcal{C}_2 where α is a primitive element of $GF(4)$:

$$H_2 = \begin{pmatrix} 1 & 1 & \alpha & \alpha^2 \\ 0 & 1 & \alpha^2 & \alpha \end{pmatrix}$$

Example of a tensor product code

- Consider the parity check matrix H_1 of a $[3, 1, 1]_2$ code \mathcal{C}_1 :

$$H_1 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

- Consider the parity check matrix H_2 of a $[4, 2, 1]_4$ code \mathcal{C}_2 where α is a primitive element of $GF(4)$:

$$H_2 = \begin{pmatrix} 1 & 1 & \alpha & \alpha^2 \\ 0 & 1 & \alpha^2 & \alpha \end{pmatrix}$$

- Map

$$0 \rightarrow \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad 1 \rightarrow \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \alpha \rightarrow \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad \alpha^2 \rightarrow \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Example of a tensor product code

- Consider the parity check matrix H_1 of a $[3, 1, 1]_2$ code \mathcal{C}_1 :

$$H_1 = \begin{pmatrix} 1 & \alpha & \alpha^2 \end{pmatrix}$$

- Consider the parity check matrix H_2 of a $[4, 2, 1]_4$ code \mathcal{C}_2 where α is a primitive element of $GF(4)$:

$$H_2 = \begin{pmatrix} 1 & 1 & \alpha & \alpha^2 \\ 0 & 1 & \alpha^2 & \alpha \end{pmatrix}$$

Example of a tensor product code

- Consider the parity check matrix H_1 of a $[3, 1, 1]_2$ code \mathcal{C}_1 :

$$H_1 = \begin{pmatrix} 1 & \alpha & \alpha^2 \end{pmatrix}$$

- Consider the parity check matrix H_2 of a $[4, 2, 1]_4$ code \mathcal{C}_2 where α is a primitive element of $GF(4)$:

$$H_2 = \begin{pmatrix} 1 & 1 & \alpha & \alpha^2 \\ 0 & 1 & \alpha^2 & \alpha \end{pmatrix}$$

- The parity check matrix of a $[1; 1]_8$ -bit-error-correcting code \mathcal{C}_A of length $n = 4$ with 3-bit symbols is:

$$H_2 \otimes H_1 = \begin{pmatrix} 1 & \alpha & \alpha^2 & 1 & \alpha & \alpha^2 & \alpha & \alpha^2 & 1 & \alpha^2 & 1 & \alpha \\ 0 & 0 & 0 & 1 & \alpha & \alpha^2 & \alpha^2 & 1 & \alpha & \alpha & \alpha^2 & 1 \end{pmatrix}.$$

Decoding algorithm

- Suppose $\mathbf{c} \in \mathcal{C}_A$ was transmitted, and $\mathbf{y} = \mathbf{c} + \mathbf{e}$ was received where \mathbf{e} is a $[t; \ell]_{2^m}$ -bit error vector.
- Two-step syndrome decoding:
 - 1 Decoder for \mathcal{C}_2 : input is the scaled syndrome \mathbf{s} of \mathbf{y} , output is decoded string \mathbf{r} .
 - 2 Decoder for \mathcal{C}_1 : input is the vector of syndromes \mathbf{r} , output is the error-vector \mathbf{e} .

Construction of a $[t_1, t_2; \ell_1, \ell_2]$ -graded bit-error-correcting code

- Suppose H_1 is the parity check matrix of a $[m, k_1, \ell_2]_2$ code \mathcal{C}_1 such that H_1 is $\begin{bmatrix} H'_1 \\ H_1 \end{bmatrix}$ where H'_1 itself is the parity check matrix of a $[m, m - r', \ell_1]_2$ code (here $r' < m - k_1$).

Construction of a $[t_1, t_2; \ell_1, \ell_2]$ -graded bit-error-correcting code

- Suppose H_1 is the parity check matrix of a $[m, k_1, \ell_2]_2$ code \mathcal{C}_1 such that H_1 is $\begin{bmatrix} H'_1 \\ H''_1 \end{bmatrix}$ where H'_1 itself is the parity check matrix of a $[m, m - r', \ell_1]_2$ code (here $r' < m - k_1$).
- H'_1 is a r' by m matrix, H''_1 is a r'' by m matrix for $r'' = r - r'$.
- One choice for H_1 is a BCH matrix.

Construction of a $[t_1, t_2; \ell_1, \ell_2]$ -graded bit error-correcting code

- Suppose H_2 is the parity check matrix of a $[n, k_2, t_1 + t_2]_{2^{r'}}$ code.

Construction of a $[t_1, t_2; \ell_1, \ell_2]$ -graded bit error-correcting code

- Suppose H_2 is the parity check matrix of a $[n, k_2, t_1 + t_2]_{2^{r'}}$ code.
- Suppose H_3 is the parity check matrix of a $[n, k_3, t_2]_{2^{r''}}$ code.

Construction of a $[t_1, t_2; \ell_1, \ell_2]$ -graded bit error-correcting code

- Suppose H_2 is the parity check matrix of a $[n, k_2, t_1 + t_2]_{2^{r'}}$ code.
- Suppose H_3 is the parity check matrix of a $[n, k_3, t_2]_{2^{r''}}$ code.

Theorem (Construction 2)

Then H_B is the parity check matrix of a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$ -graded bit error correcting code, where

$$H_B = \begin{pmatrix} H_2 \otimes H_1' \\ H_3 \otimes H_1'' \end{pmatrix}.$$

Discussion

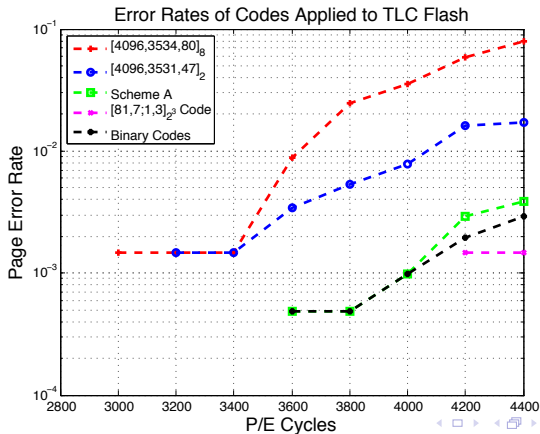
- Using sphere-packing bound argument, it follows that the excess redundancy of \mathcal{C}_B is about $t_2 \log(n)$. The code is asymptotically optimal.
- Construction 1 is also a graded-bit-error correcting code. Construction 2 offers better redundancy than Construction 1 when $(\ell_2 - \ell_1)t_1/t_2 > \log(n)/\log(m)$.
- Further simplifications are possible for special cases of the code parameters.

Evaluation

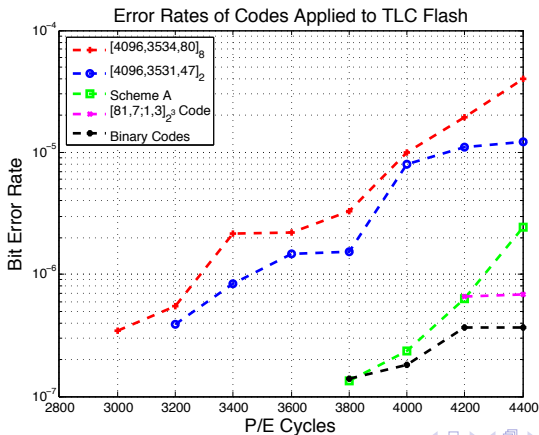
We compared the following rate-0.86 codes:

- 1 Three binary $[2^{12}, 3531, 47]_2$ codes with the same error correction capability for the LSB, CSB, and MSB pages,
- 2 Three binary codes with different error correction capability for the LSB, CSB, and MSB pages: codes $[2^{12}, 3351, 62]_2$, $[2^{12}, 3339, 63]_2$, and $[2^{12}, 3915, 15]_2$,
- 3 A non-binary $[2^{12}, 3338, 84]_4$ code over $GF(4)$ applied to the CSB and LSB sharing the same physical cells and a binary $[2^{12}, 3915, 15]_2$ code applied to the MSB (scheme A),
- 4 A non-binary $[2^{12}, 3534, 80]_8$ code over $GF(8)$ which corrects errors in a group of LSB, CSB, and MSB pages sharing the same physical cells, and
- 5 A graded bit-error-correcting $[2^{12}, 3302, 88]_4$, $[2^{12}, 4011, 7]_2$ code.

Coding can extend lifetime



Coding can extend lifetime



Issues

- Delay onset of errors
- Improve reliability
- Improve write access
- Increase storage capacity
- Reduce interference

Techniques

- Error correction codes
- Codes for rewriting data
- Rank modulation
- Constrained coding

Issues

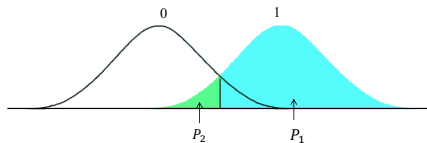
- Delay onset of errors
- Improve reliability
- Improve write access
- Increase storage capacity
- Reduce interference

Techniques

- Error correction codes
- Codes for rewriting data
- Rank modulation
- Constrained coding

Extracting soft information

- Recall: Bit-level distribution
- 1 read compares against a single threshold

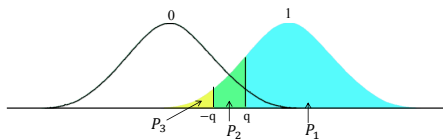


Extracting soft information

- Idea: multiple word line reads

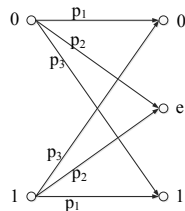
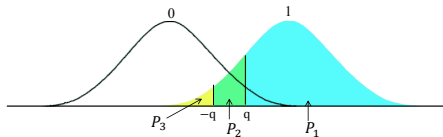
Extracting soft information

- Idea: multiple word line reads
- 2 reads compare against two thresholds



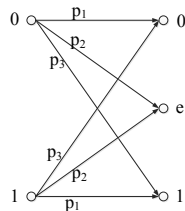
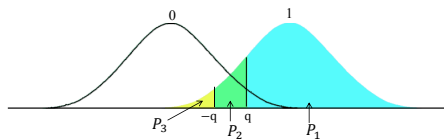
Extracting soft information

- Idea: multiple word line reads
- 2 reads compare against two thresholds



Extracting soft information

- Idea: multiple word line reads
- 2 reads compare against two thresholds



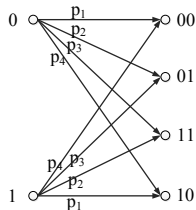
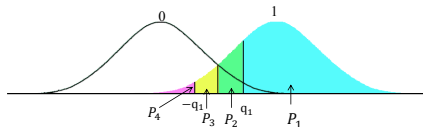
- Maximize mutual information of the induced channel to determine the best thresholds (here q and $-q$)

Extracting soft information

- Idea: multiple word line reads

Extracting soft information

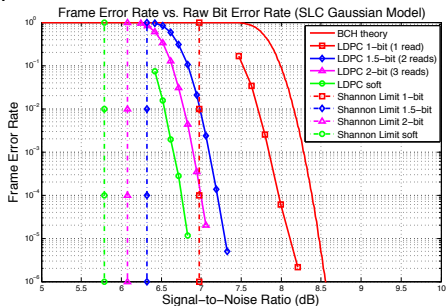
- Idea: multiple word line reads
- 3 reads compare against three thresholds



- Maximize mutual information of the induced channel to determine the best thresholds (here q_1 , $-q_1$ and 0)

Coding can improve reliability [1]

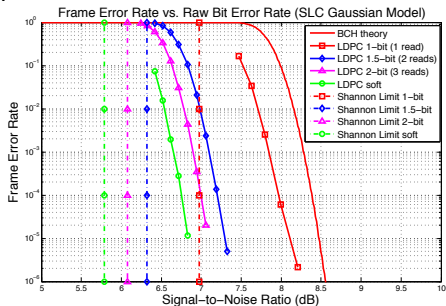
Figure: Performance comparison for 0.9-rate LDPC and BCH codes of length $n = 9100$.



[1] Wang *et al.*, "LDPC Decoding with Limited-Precision Soft Information in Flash Memories," preprint, 2012.

Coding can improve reliability [1]

Figure: Performance comparison for 0.9-rate LDPC and BCH codes of length $n = 9100$.



- Caution: AWGN-optimized LDPC codes may not be the best for the quantized Flash channel !

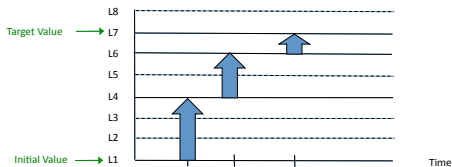
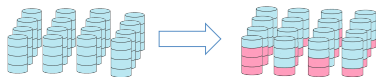
Coding can accelerate write access

- Recall Flash programming



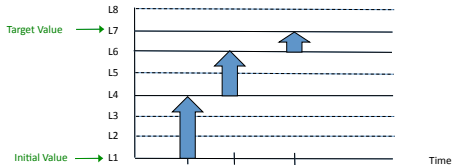
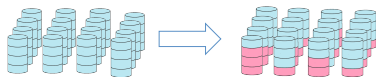
Coding can accelerate write access

- Recall Flash programming
- In practice incremental pulse programming is used, a.k.a. guess and verify.
- Latency increases with number of levels.



Coding can accelerate write access

- Recall Flash programming
- In practice incremental pulse programming is used, a.k.a. guess and verify.
- Latency increases with number of levels.
- If one allows for “dirty writes”, it suffices to correct errors in only one direction.



Asymmetric limited-magnitude error-correcting codes [1]

Definition (ALM Code)

Let \mathcal{C}_1 be a code over the alphabet Q_1 . The code \mathcal{C} over the alphabet Q (with $|Q| > |Q_1| = q_1 = \ell + 1$) is defined as $\mathcal{C} = \{\mathbf{x} = (x_1, x_2, \dots, x_n) \in Q^n \mid \mathbf{x} \bmod q_1 \in \mathcal{C}_1\}$.

Asymmetric limited-magnitude error-correcting codes [1]

Definition (ALM Code)

Let \mathcal{C}_1 be a code over the alphabet Q_1 . The code \mathcal{C} over the alphabet Q (with $|Q| > |Q_1| = q_1 = \ell + 1$) is defined as $\mathcal{C} = \{\mathbf{x} = (x_1, x_2, \dots, x_n) \in Q^n \mid \mathbf{x} \bmod q_1 \in \mathcal{C}_1\}$.

Theorem





Code \mathcal{C} corrects t asymmetric errors of limited magnitude ℓ if code \mathcal{C}_1 corrects t symmetric errors.

- New construction inherits encoding/decoding complexity of the underlying code.
- Connections with additive number theory (Varshamov 1970s).

[1] Y. Cassuto et al., "Codes for asymmetric limited-magnitude errors with application to multilevel flash memories," IEEE Trans. on Information Theory, 2010.





Further reading (1/2)

Algebraic codes

-  R. Gabrys *et al.*, “Graded bit error correcting codes with applications to flash memory,” preprint, 2012.
-  Y. Cassuto *et al.*, “Codes for asymmetric limited-magnitude errors with application to multilevel flash memories,” *IEEE Trans. on Information Theory*, 2010.
-  T. Kløve *et al.*, “Systematic, single limited magnitude error correcting codes for flash memories,” *IEEE Trans. on Information Theory*, 2011.
-  T. Kløve *et al.*, “Some codes correcting asymmetric errors of limited magnitude,” *IEEE Trans. on Information Theory*, 2011.

Further reading (2/2)

Graph-based codes

-  J. Wang *et al.*, “The Cycle Consistency Matrix Approach to LDPC Absorbing Sets in Separable Circulant-Based Codes,” preprint, 2012 (available on ArXiv).
-  J. Wang *et al.*, “LDPC Decoding with Limited-Precision Soft Information in Flash Memories,” preprint, 2012 (available on ArXiv).
-  B. Amiri *et al.*, “Quantization, Absorbing Regions and Practical Message Passing Decoders,” Asilomar 2012.
-  F. Sala *et al.*, “Dynamic Threshold Schemes for Multi-Level Nonvolatile Memories,” Asilomar 2012.

Codes for Rewriting Data

Issues

- Delay onset of errors
- Improve reliability
- Improve write access
- Increase storage capacity
- Reduce interference

Techniques

- Error correction codes
- **Codes for rewriting data**
- Rank modulation
- Constrained coding

Issues

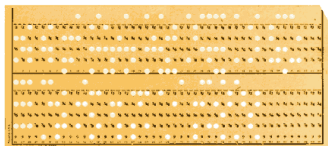
- Delay onset of errors
- Improve reliability
- Improve write access
- **Increase storage capacity**
- Reduce interference

Techniques

- Error correction codes
- **Codes for rewriting data**
- Rank modulation
- Constrained coding

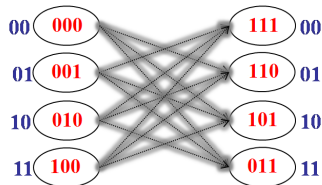
Write-once-memories (WOMs)

- The memory cells represent bits that are irreversibly programmed from “0” to “1”.
- What is the total number of bits that is possible to write over n cells in t writes?
- WOM codes were introduced by Rivest and Shamir “How to reuse a write-once memory”, *Information and Control*, 1982.
- WOM codes were originally proposed for punch cards and optical disks.



Rivest-Shamir WOM code

Information	First Generation	Second Generation
00	000	111
01	001	110
10	010	101
11	100	011



Encoding example 1

Rivest-Shamir WOM code:

Information	First Generation	Second Generation
00	000	111
01	001	110
10	010	101
11	100	011

Encoding example 1

Rivest-Shamir WOM code:

Information	First Generation	Second Generation
00	000	111
01	001	110
10	010	101
11	100	011

- Write-1: 01 → Encode: 001.

Encoding example 1

Rivest-Shamir WOM code:

Information	First Generation	Second Generation
00	000	111
01	001	110
10	010	101
11	100	011

- Write-1: 01 → Encode: 001.
- Write-2: 10 → Encode: 101.

Encoding example 1

Rivest-Shamir WOM code:

Information	First Generation	Second Generation
00	000	111
01	001	110
10	010	101
11	100	011

- Write-1: 01 → Encode: 001.
- Write-2: 10 → Encode: 101.

Encoding example 2

Rivest-Shamir WOM code:

Information	First Generation	Second Generation
00	000	111
01	001	110
10	010	101
11	100	011

Encoding example 2

Rivest-Shamir WOM code:

Information	First Generation	Second Generation
00	000	111
01	001	110
10	010	101
11	100	011

- Write-1: 01 → Encode: 001.

Encoding example 2

Rivest-Shamir WOM code:

Information	First Generation	Second Generation
00	000	111
01	001	110
10	010	101
11	100	011

- Write-1: 01 → Encode: 001.
- Write-2: 01 → Encode: 001. No change.

System model

Definition (WOM Model)

The memory state is modeled as a vector \mathbf{y}^j of length n where j is the current write (or generation). Each element y_i^j , $1 \leq i \leq n$, takes values in the set $\{0, 1, \dots, q - 1\}$. On write j , the encoder writes one of M_j messages to the memory by updating \mathbf{y}^{j-1} to \mathbf{y}^j while satisfying the **WOM-constraint** $\mathbf{y}^j \geq \mathbf{y}^{j-1}$.

System model

Definition (WOM Model)

The memory state is modeled as a vector \mathbf{y}^j of length n where j is the current write (or generation). Each element y_i^j , $1 \leq i \leq n$, takes values in the set $\{0, 1, \dots, q-1\}$. On write j , the encoder writes one of M_j messages to the memory by updating \mathbf{y}^{j-1} to \mathbf{y}^j while satisfying the **WOM-constraint** $\mathbf{y}^j \geq \mathbf{y}^{j-1}$.

Definition (Sum rate)

If M_j codewords can be represented at generation j , then generation j has rate $\frac{1}{n} \log(M_j)$. The **sum rate** is the sum of rates across generations.

WOM capacity

- Achievable rate-region computed by Fu and Han Vinck in 1999.
- The capacity (maximum achievable sum-rate) using t writes and q -ary cells is

$$\log \binom{t + q - 1}{q - 1}.$$

- Popular setting: binary 2-write WOM, the capacity is

$$\log 3 \approx 1.58.$$

- Random coding achieves capacity - the bound is tight.

Early results on binary WOM codes

Rivest-Shamir WOM code:

Information	First Generation	Second Generation
00	000	111
01	001	110
10	010	101
11	100	011

- Recall that the capacity is 1.58.
- Rivest-Shamir WOM code already achieves sum rate of $(\log 4 + \log 4)/3 = 4/3$ with $M_1 = 4$, $M_2 = 4$, and $n = 3$.

Early results on binary WOM codes

- Here is another example from Rivest and Shamir

A H G G F Y L w E Z Y r X f p n D W V z U d j o T w k e l t d u
 C S R c Q i o z P p i h u e x y O z s j s n i w v c q g f k b m
 B N M z L b g m K u t b n g f w J w r h k v x y m j p s o q c i
 I k m q l c k u w t e o s d j v u b d f g e t p y x n l h r z a

- Symbol X in position (i, j) is encoded as the number $32 \times i + j$ over $GF(2)^7$.
- First generation is in UPPER CASE.
- Example:
 - Write-1: symbol "Q" \rightarrow Encode $32 \times 1 + 4$ in binary : $(0, 1, 0, 0, 1, 0, 0)$.
 - Write-2: symbol "b" \rightarrow Encode $32 \times 1 + 30$ in binary : $(0, 1, 1, 1, 1, 1, 0)$.
- Rate ≈ 1.34 .

Early results on binary WOM codes

- Here is another example from Rivest and Shamir

A H G G F Y L w E Z Y r X f p n D W V z U d j o T w k e l t d u
 C S R c Q i o z P p i h u e x y O z s j s n i w v c q g f k b m
 B N M z L b g m K u t b n g f w J w r h k v x y m j p s o q c i
 I k m q l c k u w t e o s d j v u b d f g e t p y x n l h r z a

- Symbol X in position (i, j) is encoded as the number $32 \times i + j$ over $GF(2)^7$.
- First generation is in UPPER CASE.
- Example:
 - Write-1: symbol "Q" \rightarrow Encode $32 \times 1 + 4$ in binary :
 $(0, 1, 0, 0, 1, 0, 0)$.
 - Write-2: symbol "b" \rightarrow Encode $32 \times 1 + 30$ in binary :
 $(0, 1, 1, 1, 1, 1, 0)$.
- Rate ≈ 1.34 .

Sufficient conditions for achieving capacity

- Suppose there are q total symbols and t rewrites.


Sufficient conditions for achieving capacity

- Suppose there are q total symbols and t rewrites.
- Let $\alpha_{t-j,q,m}$ be $Pr(y_i^j = m)$ and let $Pr(y_i^{t-(j+1)} = m | y_i^{t-j} = r)$ be denoted as $\alpha_{t-i,q,m|r}$.

Sufficient conditions for achieving capacity

- Suppose there are q total symbols and t rewrites.
- Let $\alpha_{t-j,q,m}$ be $Pr(y_i^j = m)$ and let $Pr(y_i^{t-(j+1)} = m | y_i^{t-j} = r)$ be denoted as $\alpha_{t-i,q,m|r}$.
- Then, as $n \rightarrow \infty$, the symbol distributions for a random capacity achieving code are given by the expressions [1]:

$$\begin{aligned}
 \alpha_{t-0,q,0} &= \frac{t}{t-1+q} \\
 \alpha_{t-0,q,m} &= \frac{\alpha_{t-0,q,0}}{\binom{q+t-2}{t-1}} \binom{q+t-m-2}{t-1} \\
 \alpha_{t-i,q,m|r} &= \alpha_{(t-1)-(i-1),q-r,m-r}
 \end{aligned}$$


[1] F. Fu and A.J. Han Vinck, "On the capacity of generalized write-once memory with state transitions described by an arbitrary directed acyclic graph," *IEEE Trans. Inform. Theory*, 1999. 

Sufficient conditions for achieving capacity

- Suppose there are q total symbols and t rewrites.
- Let $\alpha_{t-j,q,m}$ be $Pr(y_i^j = m)$ and let $Pr(y_i^{t-(j+1)} = m | y_i^{t-j} = r)$ be denoted as $\alpha_{t-i,q,m|r}$.
- Then, as $n \rightarrow \infty$, the symbol distributions for a random capacity achieving code are given by the expressions [1]:

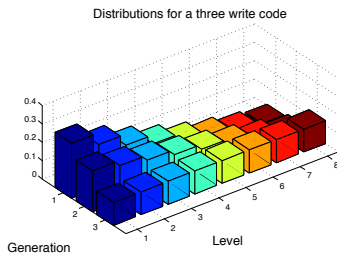
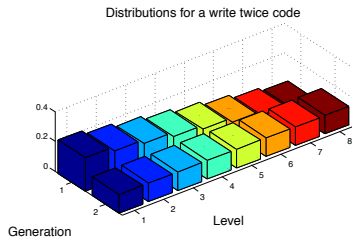
$$\begin{aligned} \alpha_{t-0,q,0} &= \frac{t}{t-1+q} \\ \alpha_{t-0,q,m} &= \frac{\alpha_{t-0,q,0}}{\binom{q+t-2}{t-1}} \binom{q+t-m-2}{t-1} \\ \alpha_{t-i,q,m|r} &= \alpha_{(t-1)-(i-1),q-r,m-r} \end{aligned}$$

We refer to the resulting code as a *random recursive (RR) code*.

[1] F. Fu and A.J. Han Vinck, "On the capacity of generalized write-once memory with state transitions described by an arbitrary directed acyclic graph," *IEEE Trans. Inform. Theory*, 1999. 

An illustration

- Distributions conditioned on the previous level.



Capacity achieving RR code exhibits regularity.

Coset Coding [1]

- Consider an error correction code \mathcal{C} with parity check matrix H .
- Construct a 2-write WOM code as follows:
 - 1 First write: Encode message m_1 into codeword c_1 such that $Hc_1 = m_1$ and c_1 is “low-weight”.
 - 2 Second write: Encode message m_2 into codeword $c_2 = c_1 + c'_2$ such that $Hc'_2 = m_1 + m_2$ and $c'_2 \not\preceq c_1$.
- “Low-weight” means less than the minimum distance. Large minimum distance implies large first generation codebook.

[1] G. D. Cohen, P. Godlewski, and F. Merks, “Linear binary code for write- once memories,” *IEEE Trans. Inform. Theory*, vol. 32, no. 5, Oct. 1986, pp. 697-700.

Rivest-Shamir code interpretation via coset coding

- Let's consider a 3-repetition code with parity check matrix

$$H = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

- On first write: $Hc_1 = m_1$

Information (m_1)	First Generation (c_1)
00	000
01	001
10	010
11	100

Rivest-Shamir code interpretation via coset coding

- Let's consider a 3-repetition code with parity check matrix

$$H = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

- On first write: $Hc_1 = m_1$

Information (m_1)	First Generation (c_1)
00	000
01	001
10	010
11	100

Rivest-Shamir code interpretation via coset coding

- Let's consider a 3-repetition code with parity check matrix

$$H = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

- On second write: Find c'_1 such that $Hc'_1 = (m_1 + m_2)$ and $c'_1 \neq c_1$.

Information (m_2)	Second Generation (c_2)
00	111
01	110
10	101
11	011

Rivest-Shamir code interpretation via coset coding

- Let's consider a 3-repetition code with parity check matrix

$$H = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

- On second write: Find c'_1 such that $Hc'_1 = (m_1 + m_2)$ and $c'_1 \neq c_1$. $Hc'_1=(11) \rightarrow c'_1=(100)$

Information (m_2)	Second Generation (c_2)
00	111
01	110
10	101
11	011

Rivest-Shamir code interpretation via coset coding

- Let's consider a 3-repetition code with parity check matrix

$$H = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

- On second write: Find c'_1 such that $Hc'_1 = (m_1 + m_2)$ and $c'_1 \neq c_1$. $Hc'_1 = (11) \rightarrow c'_1 = (100) \rightarrow c_1 + c'_1 = (001) + (100)$

Information (m_2)	Second Generation (c_2)
00	111
01	110
10	101
11	011

Recent results on binary WOM codes

- Coset coding idea was recently generalized by Wu [1] and Yaakobi *et al.* [2].
- Using first-order $[16, 5, 3]$ Reed-Muller code as the error correcting code yields sum-rate of 1.4566. (capacity is 1.58!)
- Yields a recursive construction of high-rate t -write binary WOM codes.
- Coset coding idea was further refined by Shpilka [3]: instead of one fixed matrix, choose an appropriate element from an ensemble of matrices...capacity achieving scheme!

[1] Y. Wu, Low complexity codes for writing write-once memory twice, Proc. IEEE Int. Symp. Inform. Theory (ISIT), Austin, Texas, June 2010.

[2] E. Yaakobi et al., "Codes for Write-Once Memories," IEEE Trans. on Information Theory, Sept. 2012.

[3] A. Shpilka, "New constructions of WOM codes using the Wozenkraft ensemble," LATIN, April 2012.

Constructing high-rate non-binary WOM codes

- 1 Begin with good t -write WOM codes over smaller alphabets.
- 2 Define a mapping into a code with a larger alphabet.
- 3 Construct a new non-binary code based on these constituents.

Construction I

- Consider $q = 2^m$.
- Every cell value is converted into an m -bit binary vector using binary representation.
- For $1 \leq i \leq m$, the i th bits in each cell comprise a binary t -write WOM code.
- These m binary WOM codes update values independently.
- Since each constituent code obeys the WOM constraint, resultant code also obeys the WOM constraint.

Construction I

Lemma

If $q = 2^m$ and there exists a binary t -write WOM code of sum rate R , then there exists a q -ary t -write WOM code of sum rate mR .

- Immediate extension to the case when $q = s^m$ for $s > 2$.

An example of Construction I using Rivest and Shamir write-twice code for $q = 8$ and $n = 3$

Rivest-Shamir WOM code

Information	First Generation	Second Generation
00	000	111
01	001	110
10	010	101
11	100	011

Construction I

Write no.	Data bits	Encoding by RS code	Encoded values
1	(11, 01, 10)	(100,001,010)	(4,1,2)
2	(00, 11, 01)	(101,111,110)	(5,7,6)

Construction II

- Assume $3 \mid q$ and partition the cell levels into three groups: $\{0, \dots, q/3 - 1\}$, $\{q/3, \dots, 2q/3 - 1\}$, $\{2q/3, \dots, 3q - 1\}$, each of size $q/3$.
- Let \mathcal{C} be a binary two-write WOM code of length n .
- First write – write two words:
 - A binary codeword $\mathbf{u} \in \{0, 1\}^n$ according to the first write in \mathcal{C} .
 - A message word $\mathbf{w} \in \{0, \dots, q/3 - 1\}^n$.
 - For each $i, 1 \leq i \leq n$, if $u_i = 0$ write w_i ; if $u_i = 1$ write $w_i + q/3$.
- Second write – write two words:
 - A binary codeword $\mathbf{u} \in \{0, 1\}^n$ according to the second write in \mathcal{C} .
 - A message word $\mathbf{w} \in \{0, \dots, q/3 - 1\}^n$.
 - For each $i, 1 \leq i \leq n$, if $u_i = 0$ write $w_i + q/3$; if $u_i = 1$ write $w_i + 2q/3$.

Construction II

Lemma

If $q = m(t + 1)$ and there exists a binary t -write WOM code of sum rate R , then there exists a q -ary t -write WOM code of sum rate $R + t \log(m)$.

- Immediate extension to the case when $q = m(s + t - 1)$ where there exists a s -ary t -write WOM code for $s > 2$.

An example of Construction II using Rivest and Shamir write-twice code for $q = 9$ and $n = 3$

Rivest-Shamir WOM code

Information	First Generation	Second Generation
00	000	111
01	001	110
10	010	101
11	100	011

Construction II

Write no.	Information	RS code + info	Encoded values
1	(0,1),(0,1,2)	(001),(012)	(0,1,5)
2	(0,0),(2,1,2)	(111),(212)	(8,7,8)

Results and comparison

- In general, Construction I gives better results for smaller q , Construction II gives better results for larger q .

q	Achieved sum-rate	Capacity
4	2.9856 (I)	3.3219
8	4.4784 (I)	5.1699
16	6.3083 (I)	7.0875
32	8.9684 (II)	9.0444
64	10.3083 (II)	11.0244
128	12.3083 (II)	13.0112

- Very high rate two-write non-binary WOM codes (outperform [1,2]).

[1] E. Yaakobi *et al.*, "Efficient two-write WOM codes," ITW 2010.

[2] Q. Huang, S. Lin and A. S. Abdel-Ghaffar, "Error-correcting codes for flash coding," *Trans. Info. Theory*, 2011.

Facets of WOM and rewriting codes

- Reduction of write amplification with a help of WOM code (Luijie et al., 2012)

Facets of WOM and rewriting codes

- Reduction of write amplification with a help of WOM code (Luijie et al., 2012)
- WOM codes robust to inter-cell interference (Li, 2011)

Facets of WOM and rewriting codes





- Reduction of write amplification with a help of WOM code (Luijie et al., 2012)
- WOM codes robust to inter-cell interference (Li, 2011)
- Error correction/detection in conjunction with WOM property e.g., polar codes (Burshtein and Strugatski, 2012)

Facets of WOM and rewriting codes

- Reduction of write amplification with a help of WOM code (Luijie et al., 2012)
- WOM codes robust to inter-cell interference (Li, 2011)
- Error correction/detection in conjunction with WOM property e.g., polar codes (Burshtein and Strugatski, 2012)
- Codes that maximize the number of writes between subsequent erases (cf. Floating codes invented by A. Jiang et al., 2007)





Further Reading (1/2)

Early Capacity and Code Construction Results

-  R. L. Rivest and A. Shamir, “How to reuse a “write-once” memory,” *Information and Control*, 1982.
-  J. Wolf *et al.*, “Coding for write-once memory,” *IEEE Trans. on Information Theory*, 1984.
-  C. Heegard, “On the capacity of permanent memory,” *IEEE Trans. on Information Theory*, 1985.
-  F. Fu and A. J. Han Vinck, “On the capacity of generalized write-once memory with state transitions described by an arbitrary directed acyclic graph,” *IEEE Trans. on Information Theory*, 1999.

Further Reading (2/2)

Recent Works

-  Q. Huang *et al.*, “Error-correcting codes for flash coding,” *IEEE Trans. on Information Theory*, 2011.
-  E. Yaakobi *et al.*, “Multiple error-correcting WOM-codes,” *IEEE Trans. Info. Theory*, 2012.
-  B. Kurkoski, “Asymptotic rates for lattice-based WOM codes,” *Non-Volatile Memories Workshop*, 2012.
-  R. Gabrys and L. Dolecek, “Bounds and simple constructions of non-binary write once codes for multilevel flash memories,” preprint, 2012.

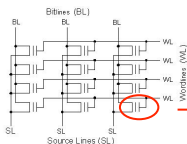
See also WOM Codes Session at IEEE ISIT 2012 as well as papers at ISIT 2007+.

Part II

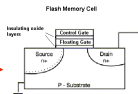
Rank Modulation

Difficulties with Flash Memories

Cell array in a flash memory



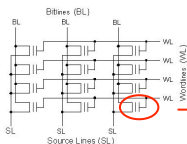
A flash memory cell
(Floating gate)



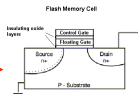
- To program (i.e., write) cells, we can only increase cell levels due to the high cost of block erasure.

Difficulties with Flash Memories

Cell array in a flash memory



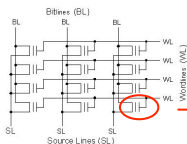
A flash memory cell (Floating gate)



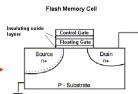
- To program (i.e., write) cells, we can only increase cell levels due to the high cost of block erasure.
- Charge injection is a noisy random process.

Difficulties with Flash Memories

Cell array in a flash memory



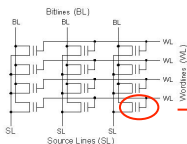
A flash memory cell (Floating gate)



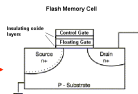
- To program (i.e., write) cells, we can only increase cell levels due to the high cost of block erasure.
- Charge injection is a noisy random process.
- Thousands of cells are programmed in parallel sharing the same programming voltage. The worse behavior of cells determines performance.

Difficulties with Flash Memories

Cell array in a flash memory



A flash memory cell (Floating gate)



- To program (i.e., write) cells, we can only increase cell levels due to the high cost of block erasure.
- Charge injection is a noisy random process.
- Thousands of cells are programmed in parallel sharing the same programming voltage. The worse behavior of cells determines performance.
- There are various mechanisms for noise: Read/Write disturbs, charge leakage, inter-cell interference, random noise.

Challenges to Flash Memories

- How to program cells reliably? Can we remove the risk of overshooting (i.e., injecting too much charge into cells)?

Challenges to Flash Memories

- How to program cells reliably? Can we remove the risk of overshooting (i.e., injecting too much charge into cells)?
- When errors appear, can we physically correct cell levels (instead of just finding out what the errors are via ECC decoding) without block erasures?

Challenges to Flash Memories

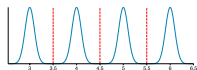
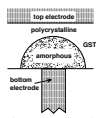
- How to program cells reliably? Can we remove the risk of overshooting (i.e., injecting too much charge into cells)?
- When errors appear, can we physically correct cell levels (instead of just finding out what the errors are via ECC decoding) without block erasures?
- Can we rewrite data *easily* without block erasures?

Challenges to Flash Memories

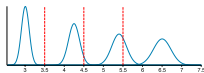
- How to program cells reliably? Can we remove the risk of overshooting (i.e., injecting too much charge into cells)?
- When errors appear, can we physically correct cell levels (instead of just finding out what the errors are via ECC decoding) without block erasures?
- Can we rewrite data *easily* without block erasures?
- Can we balance the cell levels in the same page by adaptively setting the gaps between adjacent cell levels, during writing and rewriting?

Difficulties and Challenges to Phase Change Memories

- Increasing and decreasing cell levels have different cost.
- The change of cell levels during programming is a random process.
- Thousands of cells are programmed in parallel sharing the same programming voltage. The worst behavior of cells determines performance.
- Cell levels drift toward the amorphous state after programming, which is a serious problem.
- How to program cells reliably, fast, and adaptively?



(a) Distributions before drift



(b) Distributions after drift

Definition of Rank Modulation [1-2]

Rank Modulation:

We use the relative order of cell levels (instead of their absolute values) to represent data.



[1] A. Jiang, R. Mateescu, M. Schwartz and J. Bruck, "Rank Modulation for Flash Memories," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, pp. 1731–1735, July 2008.

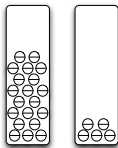
[2] A. Jiang, M. Schwartz and J. Bruck, "Error-Correcting Codes for Rank Modulation," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, pp. 1736–1740, July 2008.

Examples and Extensions of Rank Modulation

- Example: Use 2 cells to store 1 bit.

Relative order: (1,2)

Value of data: 0

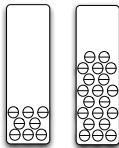


cell 1

cell 2

Relative order: (2,1)

Value of data: 1

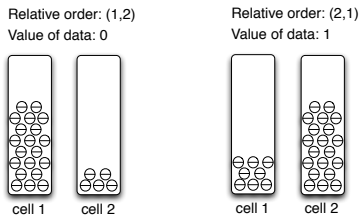


cell 1

cell 2

Examples and Extensions of Rank Modulation

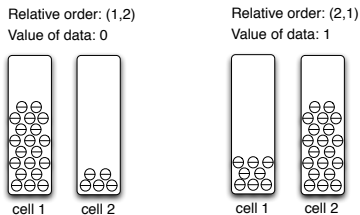
- Example: Use 2 cells to store 1 bit.



- Example: Use 3 cells to store $\log_2 6$ bits. The relative orders $(1, 2, 3), (1, 3, 2), \dots, (3, 2, 1)$ are mapped to data $0, 1, \dots, 5$.

Examples and Extensions of Rank Modulation

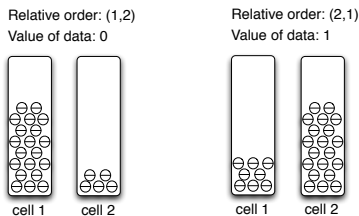
- Example: Use 2 cells to store 1 bit.



- Example: Use 3 cells to store $\log_2 6$ bits. The relative orders $(1, 2, 3), (1, 3, 2), \dots, (3, 2, 1)$ are mapped to data $0, 1, \dots, 5$.
- In general, k cells can represent $\log_2(k!)$ bits.

Examples and Extensions of Rank Modulation

- Example: Use 2 cells to store 1 bit.



- Example: Use 3 cells to store $\log_2 6$ bits. The relative orders $(1, 2, 3), (1, 3, 2), \dots, (3, 2, 1)$ are mapped to data $0, 1, \dots, 5$.
- In general, k cells can represent $\log_2(k!)$ bits.
- Extensions: (1) We can partition cells into groups, and apply rank modulation to each group. (2) We can let each rank contain multiple cells, namely, to extend permutations to multi-set permutations.

Advantages of Rank Modulation

- Reliable and fast cell programming: No risk of overshooting.
- More tolerance of asymmetric noise and cell-level drifting.
- Ability to remove errors and balance cell levels by physically adjusting cell levels: Only the relative order of the cell levels matters, so we can adaptively increase cell levels.

In the following, we consider “Codes for Rewriting” and “Error-Correcting Codes”, in the framework of flash memories. (But many of the concepts can be applied to phase-change memories, too.)

Rewriting Codes for Rank Modulation

Issues

- Delay onset of errors
- Improve reliability
- Improve write access
- Increase storage capacity
- Reduce interference

Techniques

- Error correction codes
- Codes for rewriting data
- Rank modulation (with rewriting)
- Constrained coding

Rewriting Codes for Rank Modulation

Issues

- Delay onset of errors
- Improve reliability
- **Improve write access**
- **Increase storage capacity**
- Reduce interference

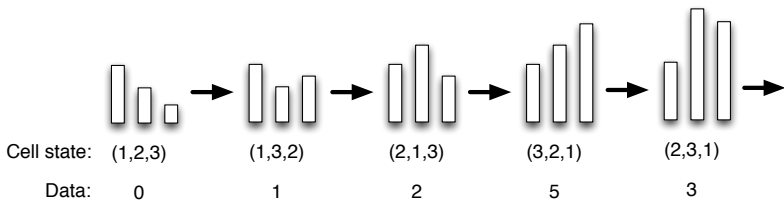
Techniques

- Error correction codes
- Codes for rewriting data
- **Rank modulation (with rewriting)**
- Constrained coding

Rewriting Codes for Rank Modulation

We can change the relative order of cell levels by only increasing cell levels, and therefore rewrite data.

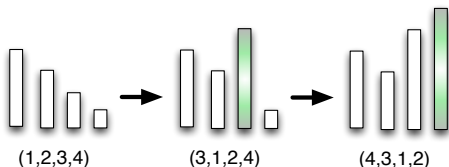
Example: Use 3 cells to store data of 6 possible values: $0, 1, \dots, 5$.
 When the data change as $0 \rightarrow 1 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow \dots$, the cell state changes as:



Prefix Codes with Push-to-Top Operation

Push-to-Top operation: Each time, we push one cell's level to be higher than all the other cells' levels. (It is easy to implement.)

Example: Say we need to change the permutation from $(1, 2, 3, 4)$ to $(4, 3, 1, 2)$.



Cost of Rewriting: The number of cells that are pushed. (It represents by how much the highest cell level is increased.)

Prefix Codes with Push-to-Top Operation

Prefix Code: We use the prefixes of the permutations to represent data. Those prefixes cannot be prefixes of each other.

Prefix Codes with Push-to-Top Operation

Prefix Code: We use the prefixes of the permutations to represent data. Those prefixes cannot be prefixes of each other.

Intuitive reason: When we push cells to the top, we are changing the prefix of the permutation.

Prefix Codes with Push-to-Top Operation

Prefix Code: We use the prefixes of the permutations to represent data. Those prefixes cannot be prefixes of each other.

Intuitive reason: When we push cells to the top, we are changing the prefix of the permutation.

Example: We use 3 cells to represent data of 6 values. Use prefixes of length 2: (1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1).

The maximum rewriting cost is 2.

Prefix Codes with Push-to-Top Operation

Prefix Code: We use the prefixes of the permutations to represent data. Those prefixes cannot be prefixes of each other.

Intuitive reason: When we push cells to the top, we are changing the prefix of the permutation.

Example: We use 3 cells to represent data of 6 values. Use prefixes of length 2: (1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1).

The maximum rewriting cost is 2.

Example: Use 3 cells to represent data of 3 values. Use prefixes of length 1: (1,*,*), (2,*,*), (3,*,*). Maximum rewriting cost is 1.

Prefix Codes with Push-to-Top Operation

Prefix Code: We use the prefixes of the permutations to represent data. Those prefixes cannot be prefixes of each other.

Intuitive reason: When we push cells to the top, we are changing the prefix of the permutation.

Example: We use 3 cells to represent data of 6 values. Use prefixes of length 2: (1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1).

The maximum rewriting cost is 2.

Example: Use 3 cells to represent data of 3 values. Use prefixes of length 1: (1,*,*), (2,*,*), (3,*,*). Maximum rewriting cost is 1.

There is a tradeoff between code rate and rewriting cost.

State Diagram for Permutations

State Diagram: Vertices are permutations. There is a directed edge (u, v) from u to v if the cost of changing u to v is 1, with the push-to-top operation.

State Diagram for Permutations

State Diagram: Vertices are permutations. There is a directed edge (u, v) from u to v if the cost of changing u to v is 1, with the push-to-top operation.

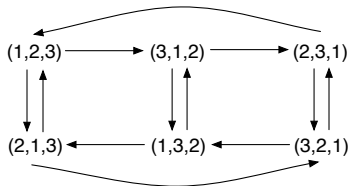
In general, the cost of changing u to v equals the length of the shortest path from u to v in the state graph.

State Diagram for Permutations

State Diagram: Vertices are permutations. There is a directed edge (u, v) from u to v if the cost of changing u to v is 1, with the push-to-top operation.

In general, the cost of changing u to v equals the length of the shortest path from u to v in the state graph.

Example: 3 cells. The state diagram is:



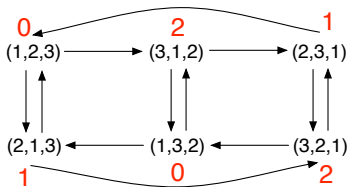
Mapping Permutations to Data Values

By mapping permutations to data values appropriately (with dominating set type of partitions in the state diagram), we can minimize the maximum rewriting cost.

Mapping Permutations to Data Values

By mapping permutations to data values appropriately (with dominating set type of partitions in the state diagram), we can minimize the maximum rewriting cost.

Example: Use 3 cells to store data of 3 values. With the following mapping, the maximum rewriting cost is 1.



Basic Concepts

n : The number of cells.

S_n : The set of $n!$ permutations.

$d(u, v)$: The distance from $u \in S_n$ to $v \in S_n$ in the state graph.

We define the ball $B_r(u)$ as $B_r(u) = \{v \in S_n \mid d(u, v) \leq r\}$.

V : The set of values of the stored data.

The rate of the code is define as $\frac{\log_2 |V|}{\log_2 (n!)}$.

Since for any $u, v \in S_n$, $|B_r(u)| = |B_r(v)|$, let $|B_r| \triangleq |B_r(u)|$.

Fact 1: $|B_r| = \frac{n!}{(n-r)!}$, which equals the number of prefixes of length r .

Fact 2: If maximum rewrite cost = r , code rate $\leq \frac{\log_2 |B_r|}{\log_2 (n!)}$.

Optimal Code That Minimizes Maximum Rewrite Cost

Given n and $|V|$, how to build a rewriting code that minimizes the maximum (i.e., worst-case) rewrite cost?

Let ρ be the smallest integer such that $|B_\rho| \geq |V|$.

Fact: The maximum rewrite cost is at least ρ .

Optimal Code That Minimizes Maximum Rewrite Cost

Given n and $|V|$, how to build a rewriting code that minimizes the maximum (i.e., worst-case) rewrite cost?

Let ρ be the smallest integer such that $|B_\rho| \geq |V|$.

Fact: The maximum rewrite cost is at least ρ .

Construction (Optimal Prefix Code)

Choose $|V|$ prefixes (of permutations) of length ρ . Map permutations of the same prefix to the same data value.

Optimal Code That Minimizes Maximum Rewrite Cost

Example: Say we are given 4 cells to store $\log_2 10$ bits. Since there are only 4 prefixes of length 1 but $12 \geq 10$ prefixes of length 2, we should choose prefixes of length 2, and the maximum rewrite cost is 2.

We can map prefixes to data values as follows:

Prefix	(1,2)	(1,3)	(1,4)	(2,1)	(2,3)
Data	0	1	2	3	4
Prefix	(2,4)	(3,1)	(3,2)	(3,4)	(4,1)
Data	5	6	7	8	9

All permutations with the same prefix represent the same data. For example, (1, 2, 3, 4) and (1, 2, 4, 3) represent the same data 0.

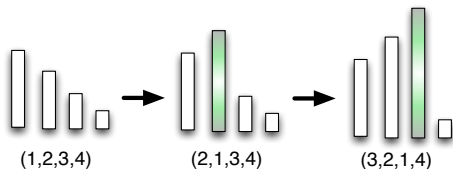
Optimal Code That Minimizes Maximum Rewrite Cost

Example (continued): Now consider rewriting. Say the current cell state (permutation) is $(1, 2, 3, 4)$, which represents data 0. We want to change the data to 7. How to do it?

Optimal Code That Minimizes Maximum Rewrite Cost

Example (continued): Now consider rewriting. Say the current cell state (permutation) is $(1, 2, 3, 4)$, which represents data 0. We want to change the data to 7. How to do it?

Solution: Choose a permutation such that: (1) Its prefix represents 7; (2) The remaining numbers have the same order as in the original permutation $(1, 2, 3, 4)$. So we choose $(3, 2, 1, 4)$ as the new permutation. Then the process of cell programming with push-to-top operations is as follows:



Rewriting Codes for Average-case Performance

Model: The sequence of rewritten data are i.i.d., with a known distribution. We want to minimize the expected cost of rewriting.

Rewriting Codes for Average-case Performance

Model: The sequence of rewritten data are i.i.d., with a known distribution. We want to minimize the expected cost of rewriting. Let n denote the number of cells, and $|V|$ denote the number of data values. We will present a prefix code with the following performance:

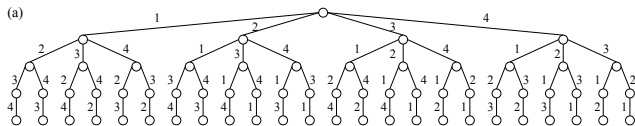
Construction (Prefix Code for Expected Rewriting Cost)

We present a prefix code with this performance: For every rewrite (not just for a sequence of rewrites), when $|V| \leq \frac{n!}{2}$, its expected rewrite cost is at most 3 times the optimal expected cost; when $n \geq 4$ and $|V| \leq \frac{n!}{6}$, it is at most 2 times the optimal.

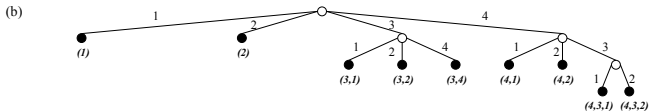
Rewriting Codes for Average-case Performance

Observation: The $n!$ permutations can be represented by a tree. Their prefix-free prefixes can be represented by the leaves of a subtree.

Example: The tree for permutations of length $n = 4$ is:



And the following subtree represents a prefix code of 9 codewords:



Rewriting Codes for Average-case Performance [1]

Construction (Prefix Code for Expected Rewriting Cost)

Build a prefix code that minimizes the expected length of the codewords (i.e., prefixes). The code construction has time complexity $O(n|V|^4)$, and achieves approximation ratios as introduced previously (compared to any code, not just prefix code).

Rewriting Codes for Average-case Performance [1]

Construction (Prefix Code for Expected Rewriting Cost)

Build a prefix code that minimizes the expected length of the codewords (i.e., prefixes). The code construction has time complexity $O(n|V|^4)$, and achieves approximation ratios as introduced previously (compared to any code, not just prefix code).

Example: Let $n = 4$, $|V| = 9$. Let the probabilities and a code be as follows. Then the expected codeword length is:

$$p_0 + p_1 + 2p_2 + 2p_3 + 2p_4 + 2p_5 + 2p_6 + 3p_7 + 3p_8.$$

Data	0	1	2	3	4	5	6	7	8
Probability	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
Codewords	(1)	(2)	(3, 1)	(3, 2)	(3, 4)	(4, 1)	(4, 2)	(4, 3, 1)	(4, 3, 2)

[1] A. Jiang, R. Mateescu, M. Schwartz and J. Bruck, "Rank Modulation for Flash Memories," in *IEEE Transactions on Information Theory*, vol. 55, no. 6, pp. 2659–2673, June 2009.

Rewriting Codes for Average-case Performance

Idea of Proof: Consider a rewrite. Ideally, which code will minimize the expected rewrite cost for *this* rewrite?

Assume the data before rewriting is $i \in \{0, 1, \dots, |V| - 1\}$. Without loss of generality, assume

$$p_0 \geq p_1 \geq \dots \geq p_{|V|-1}.$$

Then the following “ideal code” minimizes the expected rewrite cost for this rewrite:

- Let $u \in S_n$ denote the permutation before the rewrite. Sort the $n!$ permutations as $v_0, v_1, \dots, v_{n!-1}$ such that the rewrite cost

$$d(u, v_0) \leq d(u, v_1) \leq \dots \leq d(u, v_{n!-1}).$$

Map $v_0, v_1, \dots, v_{|V|-1}$ to data $i, 0, 1, \dots, i - 1, i + 1, \dots, |V| - 1$, respectively.

Rewriting Codes for Average-case Performance

Idea of Proof (continued): Consider the case $|V| \leq \frac{n!}{2}$. By the Kraft-McMillan Inequality, there exists a subtree of the tree for permutations, where the $|V|$ data values are mapped to its leaves, such that:

- The codeword for data i has length 1.
- For every other data value $i \in \{0, 1, \dots, |V| - 1\} - \{i\}$, its codeword's length is at most 3 times of its corresponding “ideal codeword length”.

This subtree is a “specific code” for this rewrite.

Rewriting Codes for Average-case Performance

Idea of Proof (continued): We have:

- Expected rewrite cost of the constructed prefix code
 \leq Expected codeword length of the constructed prefix code, but excluding data i
 \leq Expected codeword length of the “specific prefix code” in the previous slide, but excluding data i
 $\leq 3 \times$ Expected rewrite cost of the “ideal code.”
- When $n \geq 4$ and $|V| \leq \frac{n!}{6}$, similar proof applies.

Rewriting Codes with Minimal-Push-Up Operation [1]

Push-to-top is simple, but not necessary optimal (in minimizing rewrite cost).

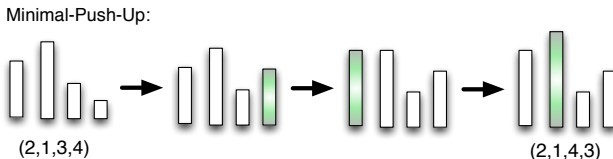
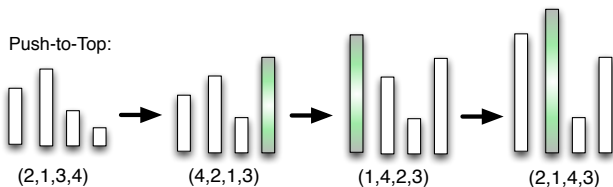
Minimal-Push-Up operation: Each time, we push one cell's level to be higher than those cells that should be below it in the final permutation (instead of higher than all the other cells).

Note: For both push-to-top and minimal-push-up, cells are pushed in the same order. The difference is by how much cell levels are pushed.

[1] E. En Gad, A. Jiang and J. Bruck, "Compressed Encoding for Rank Modulation," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, pp. 849–853, St. Petersburg, Russia, August 2011.

Rewriting Codes with Minimal-Push-Up Operation

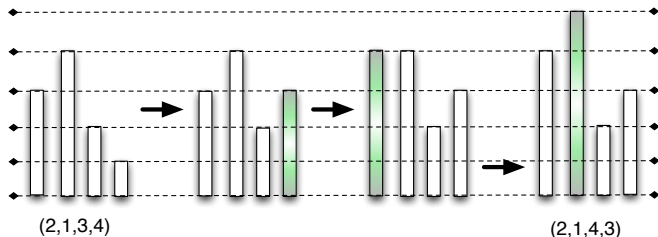
Example: We want to change the permutation from $(2, 1, 3, 4)$ to $(2, 1, 4, 3)$.



Rewriting Codes with Minimal-Push-Up Operation

To measure rewrite cost, we use discrete “virtual levels.”

Example: Changing $(2, 1, 3, 4)$ to $(2, 1, 4, 3)$.



Since highest virtual level increases by 1, rewrite cost = 1.

Rewriting Codes with Minimal-Push-Up Operation

Definition (Ball of Radius r)

Given $u \in S_n$ and non-negative integer r , let $B_r(u)$ be the set of permutations such that changing u to them has rewrite cost at most r .

- With push-to-top operations, $|B_r(u)| = \frac{n!}{(n-r)!}$.
Example: When $r = 1$, $|B_r(u)| = n$.
- With minimal-push-up, $|B_r(u)| = r!(r+1)^{n-r}$.
Example: When $r = 1$, $|B_r(u)| = 2^{n-1}$.

So minimal-push-up *can* improve performance significantly.

Example of Specific Rewriting Code with Minimal-Push-Up Operation [1]

Given Parameters: Use $n = 4$ cells to store data of $|V| = 6$ values, and maximum rewrite cost = 1.

In the rewrite code, the mapping from permutations to data is:

Permutations	(1,2,3,4)	(1,2,4,3)	(1,3,2,4)	(1,3,4,2)	(1,4,2,3)	(1,4,3,2)
	(2,3,4,1)	(2,4,3,1)	(3,2,4,1)	(3,4,2,1)	(4,2,3,1)	(4,3,2,1)
	(3,4,1,2)	(4,3,1,2)	(2,4,1,3)	(4,2,1,3)	(2,3,1,4)	(3,2,1,4)
	(4,1,2,3)	(3,1,2,4)	(4,1,3,2)	(2,1,3,4)	(3,1,4,2)	(2,1,4,3)
Data	0	1	2	3	4	5

Proof of correctness: Each coset of permutations is a dominating set.

Example of Specific Rewriting Code with Minimal-Push-Up Operation [1]

Given Parameters: Use $n = 4$ cells to store data of $|V| = 6$ values, and maximum rewrite cost = 1.

In the rewrite code, the mapping from permutations to data is:

Permutations	(1,2,3,4)	(1,2,4,3)	(1,3,2,4)	(1,3,4,2)	(1,4,2,3)	(1,4,3,2)
	(2,3,4,1)	(2,4,3,1)	(3,2,4,1)	(3,4,2,1)	(4,2,3,1)	(4,3,2,1)
	(3,4,1,2)	(4,3,1,2)	(2,4,1,3)	(4,2,1,3)	(2,3,1,4)	(3,2,1,4)
	(4,1,2,3)	(3,1,2,4)	(4,1,3,2)	(2,1,3,4)	(3,1,4,2)	(2,1,4,3)
Data	0	1	2	3	4	5






Proof of correctness: Each coset of permutations is a dominating set.

In comparison, with push-to-top operations, when $n = 4$ and $r = 1$, the stored data can have at most 4 values. So minimal-push-up can achieve higher rate. But currently, relatively less is known about its codes than push-to-top coding.





[1] E. En Gad, A. Jiang and J. Bruck, "Compressed Encoding for Rank Modulation," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, pp. 849–853, St. Petersburg, Russia, August 2011.

Further reading (1/3)

Rank Modulation and Its Rewriting Codes

-  A. Jiang, R. Matescu, M. Schwartz and J. Bruck, “Rank Modulation for Flash Memories,” *Proc. ISIT*, pp. 1731–1735, 2008.
-  A. Jiang, R. Matescu, M. Schwartz and J. Bruck, “Rank Modulation for Flash Memories,” *IEEE Transactions on Information Theory*, vol. 55, no. 6, pp. 2659–2673, 2009.
-  Z. Wang, A. Jiang and J. Bruck, “On the Capacity of Bounded Rank Modulation for Flash Memories,” *Proc. ISIT*, pp. 1234–1238, 2009.
-  Z. Wang and J. Bruck, “Partial Rank Modulation for Flash Memories,” *Proc. ISIT*, pp. 864–868, 2010.
-  M. Schwartz, “Constant-Weight Gray Codes for Local Rank Modulation,” *Proc. ISIT*, pp. 869–873, 2010.

Further reading (2/3)

-  E. En Gad, M. Langberg, M. Schwartz and J. Bruck, “On a Construction for Constant-weight Gray Codes for Local Rank Modulation,” *Proc. IEEE*, p. 996, 2010.
-  A. Jiang and Y. Wang, “Rank Modulation with Multiplicity,” *Proc. Globecom 2010 Workshop on Application of Communication Theory to Emerging Memory Technologies*, pp. 1928–1932, 2010.
-  I. Tamo and M. Schwartz, “On Optimal Anticodes over Permutations with the Infinity Norm,” *Proc. ITA Workshop*, 2011.
-  E. En Gad, A. Jiang and J. Bruck, “Compressed Encoding for Rank Modulation,” *Proc. ISIT*, pp. 849–853, 2011.

Further reading (3/3)

-  E. En Gad, M. Langberg, M. Schwartz and J. Bruck, “Generalized Gray Codes for Local Rank Modulation,” *Proc. ISIT*, pp. 839–843, 2011.
-  E. En Gad, M. Langberg, M. Schwartz and J. Bruck, “Constant-weight Gray Codes for Local Rank Modulation,” *IEEE Transactions on Information Theory*, vol. 57, no. 11, pp. 7431–7442, 2011.
-  E. En Gad, A. Jiang and J. Bruck, “Trade-offs between Instantaneous and Total Capacity in Multi-Cell Flash Memories,” *Proc. ISIT*, 2012.
-  M. Kim, J. K. Park and C. M. Twigg, “Rank Modulation Hardware for Flash Memories,” *Proc. MWSCAS*, pp. 294–297, 2012.

Error-Correcting Codes for Rank Modulation

Issues

- Delay onset of errors
- Improve reliability
- Improve write access
- Increase storage capacity
- Reduce interference

Techniques

- Error correction codes
- Codes for rewriting data
- Rank modulation (with error correction)
- Constrained coding

Error-Correcting Codes for Rank Modulation

Issues

- Delay onset of errors
- Improve reliability
- Improve write access
- Increase storage capacity
- Reduce interference

Techniques

- Error correction codes
- Codes for rewriting data
- Rank modulation (with error correction)
- Constrained coding

Error Models and Distance between Permutations

Based on the error model, there are various reasonable choices for the distance between permutations:

- Kendall-tau distance. (To be introduced in detail.)
- L_∞ distance.
- Gaussian noise based distance.
- Distance defined based on asymmetric errors or inter-cell interference.

We should choose the distance appropriately based on the type and magnitude of errors.

Kendall-tau Distance for Rank Modulation ECC [1]

When errors happen, the smallest change in a permutation is the local exchange of two adjacent numbers in the permutation. That is,

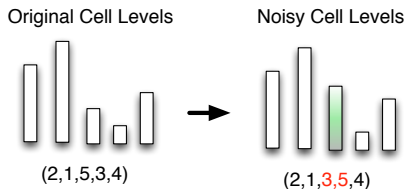
$$(a_1, \dots, a_{i-1}, a_i, a_{i+1}, a_{i+2}, \dots, a_n) \rightarrow (a_1, \dots, a_{i-1}, a_{i+1}, a_i, a_{i+2}, \dots, a_n)$$

Kendall-tau Distance for Rank Modulation ECC [1]

When errors happen, the smallest change in a permutation is the local exchange of two adjacent numbers in the permutation. That is,

$$(a_1, \dots, a_{i-1}, a_i, a_{i+1}, a_{i+2}, \dots, a_n) \rightarrow (a_1, \dots, a_{i-1}, a_{i+1}, a_i, a_{i+2}, \dots, a_n)$$

Example:

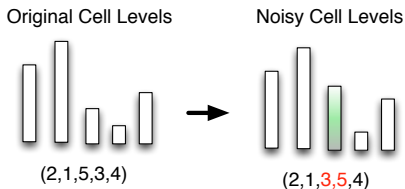


Kendall-tau Distance for Rank Modulation ECC [1]

When errors happen, the smallest change in a permutation is the local exchange of two adjacent numbers in the permutation. That is,

$$(a_1, \dots, a_{i-1}, a_i, a_{i+1}, a_{i+2}, \dots, a_n) \rightarrow (a_1, \dots, a_{i-1}, a_{i+1}, a_i, a_{i+2}, \dots, a_n)$$

Example:



We can extend the concept to multiple such “local exchanges” (for larger errors).

Kendall-tau Distance for Rank Modulation ECC

Definition (Adjacent Transposition)

An adjacent transposition is the local exchange of two neighboring numbers in a permutation, namely,

$$(a_1, \dots, a_{i-1}, a_i, a_{i+1}, a_{i+2}, \dots, a_n) \rightarrow (a_1, \dots, a_{i-1}, a_{i+1}, a_i, a_{i+2}, \dots, a_n)$$

Kendall-tau Distance for Rank Modulation ECC

Definition (Adjacent Transposition)

An adjacent transposition is the local exchange of two neighboring numbers in a permutation, namely,

$$(a_1, \dots, a_{i-1}, a_i, a_{i+1}, a_{i+2}, \dots, a_n) \rightarrow (a_1, \dots, a_{i-1}, a_{i+1}, a_i, a_{i+2}, \dots, a_n)$$

Definition (Kendall-tau Distance)

Given two permutations A and B , the Kendall-tau distance between them, $d_\tau(A, B)$, is the minimum number of adjacent transpositions needed to change A into B . (Note that $d_\tau(A, B) = d_\tau(B, A)$.)

Kendall-tau Distance for Rank Modulation ECC

Definition (Adjacent Transposition)

An adjacent transposition is the local exchange of two neighboring numbers in a permutation, namely,

$$(a_1, \dots, a_{i-1}, a_i, a_{i+1}, a_{i+2}, \dots, a_n) \rightarrow (a_1, \dots, a_{i-1}, a_{i+1}, a_i, a_{i+2}, \dots, a_n)$$

Definition (Kendall-tau Distance)

Given two permutations A and B , the Kendall-tau distance between them, $d_\tau(A, B)$, is the minimum number of adjacent transpositions needed to change A into B . (Note that $d_\tau(A, B) = d_\tau(B, A)$.)

If the minimum Kendall-tau distance of a code is $2t+1$, then it can correct t adjacent transposition errors.

Kendall-tau Distance for Rank Modulation ECC

Example: Let $A = (2, 1, 3, 4)$ and $B = (2, 3, 4, 1)$. Then $d_\tau(A, B) = 2$, and the transition from A to B is

$$A = (2, 1, 3, 4) \rightarrow (2, \underline{3}, 1, 4) \rightarrow (2, 3, \underline{4}, 1) = B.$$

Kendall-tau Distance for Rank Modulation ECC

Example: Let $A = (2, 1, 3, 4)$ and $B = (2, 3, 4, 1)$. Then $d_\tau(A, B) = 2$, and the transition from A to B is

$$A = (2, 1, 3, 4) \rightarrow (2, \underline{3}, 1, 4) \rightarrow (2, 3, \underline{4}, 1) = B.$$

Fact: For two permutations $A, B \in S_n$, $d_\tau(A, B) \leq \binom{n}{2}$.

Kendall-tau Distance for Rank Modulation ECC

Example: Let $A = (2, 1, 3, 4)$ and $B = (2, 3, 4, 1)$. Then $d_\tau(A, B) = 2$, and the transition from A to B is

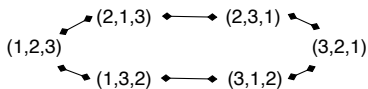
$$A = (2, 1, 3, 4) \rightarrow (2, \underline{3}, 1, 4) \rightarrow (2, 3, \underline{4}, 1) = B.$$

Fact: For two permutations $A, B \in S_n$, $d_\tau(A, B) \leq \binom{n}{2}$.

Definition (State Diagram)

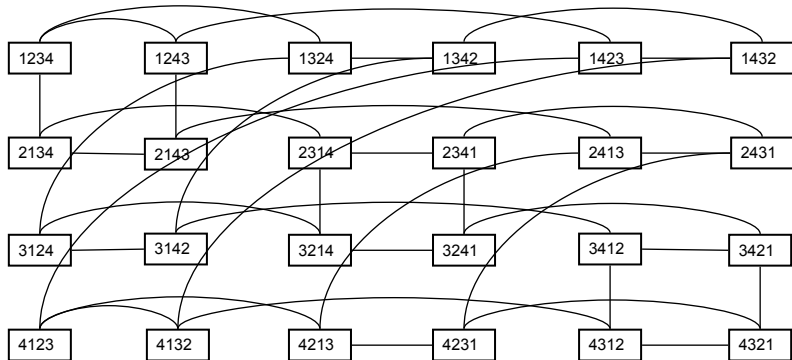
Vertices are permutations. There is an undirected edge between two permutations $A, B \in S_n$ iff $d_\tau(A, B) = 1$.

Example: The state diagram for $n = 3$ cells is



Kendall-tau Distance for Rank Modulation ECC

Example: The state diagram for $n = 4$ cells is



One-Error-Correcting Code

We introduce an error-correcting code of minimum Kendall-tau distance 3, which corrects one Kendall (i.e., adjacent transposition) error.

One-Error-Correcting Code

We introduce an error-correcting code of minimum Kendall-tau distance 3, which corrects one Kendall (i.e., adjacent transposition) error.

Definition (Inversion Vector)

Given a permutation (a_1, a_2, \dots, a_n) , its inversion vector $(x_1, x_2, \dots, x_{n-1}) \in \{0, 1\} \times \{0, 1, 2\} \times \dots \times \{0, 1, \dots, n-1\}$ is determined as follows:

- For $i = 1, 2, \dots, n-1$, x_i is the number of elements in $\{1, 2, \dots, i\}$ that are behind $i+1$ in the permutation (a_1, \dots, a_n) .

One-Error-Correcting Code

We introduce an error-correcting code of minimum Kendall-tau distance 3, which corrects one Kendall (i.e., adjacent transposition) error.

Definition (Inversion Vector)

Given a permutation (a_1, a_2, \dots, a_n) , its inversion vector $(x_1, x_2, \dots, x_{n-1}) \in \{0, 1\} \times \{0, 1, 2\} \times \dots \times \{0, 1, \dots, n-1\}$ is determined as follows:

- For $i = 1, 2, \dots, n-1$, x_i is the number of elements in $\{1, 2, \dots, i\}$ that are behind $i+1$ in the permutation (a_1, \dots, a_n) .

Example: The inversion vector for $(1, 2, 3, 4)$ is $(0, 0, 0)$. The inversion for $(4, 3, 2, 1)$ is $(1, 2, 3)$. The inversion vector for $(2, 4, 3, 1)$ is $(1, 1, 2)$.

One-Error-Correcting Code [1]

By viewing the inversion vector as coordinates, we embed permutations in an $(n - 1)$ -dimensional space.

One-Error-Correcting Code [1]

By viewing the inversion vector as coordinates, we embed permutations in an $(n - 1)$ -dimensional space.

Fact: For any two permutations $A, B \in S_n$, $d_\tau(A, B)$ is no less than their L_1 distance in the $(n - 1)$ -dimensional space.

One-Error-Correcting Code [1]

By viewing the inversion vector as coordinates, we embed permutations in an $(n - 1)$ -dimensional space.

Fact: For any two permutations $A, B \in S_n$, $d_\tau(A, B)$ is no less than their L_1 distance in the $(n - 1)$ -dimensional space.

Idea: We can construct a code of minimum L_1 distance D in the $(n - 1)$ -dimensional array of size $2 \times 3 \times \cdots \times n$. Then it is a code of Kendall-tau distance at least D for the permutations.

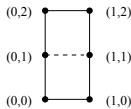
[1] A. Jiang, M. Schwartz and J. Bruck, "Error-Correcting Codes for Rank Modulation," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, pp. 1736–1740, July 2008.

One-Error-Correcting Code

Example: When $n = 3$ or $n = 4$, the embedding is as follows. (Only the solid edges are the edges in the state graph of permutations.)

Permutation	Coordinates
1 2 3	→ (0,0)
1 3 2	→ (0,1)
2 1 3	→ (1,0)
2 3 1	→ (1,1)
3 1 2	→ (0,2)
3 2 1	→ (1,2)

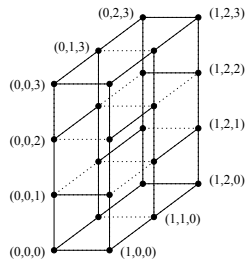
(a)



(b)

Permutation	Coordinates	Permutation	Coordinates
1 2 3 4	→ (0,0,0)	3 1 2 4	→ (0,2,0)
1 2 4 3	→ (0,0,1)	3 1 4 2	→ (0,2,1)
1 3 2 4	→ (0,1,0)	3 2 1 4	→ (1,2,0)
1 3 4 2	→ (0,1,1)	3 2 4 1	→ (1,2,1)
1 4 2 3	→ (0,0,2)	3 4 1 2	→ (0,2,2)
1 4 3 2	→ (0,1,2)	3 4 2 1	→ (1,2,2)
2 1 3 4	→ (1,0,0)	4 1 2 3	→ (0,0,3)
2 1 4 3	→ (1,0,1)	4 1 3 2	→ (0,1,3)
2 3 1 4	→ (1,1,0)	4 2 1 3	→ (1,0,3)
2 3 4 1	→ (1,1,1)	4 2 3 1	→ (1,1,3)
2 4 1 3	→ (1,0,2)	4 3 1 2	→ (0,2,3)
2 4 3 1	→ (1,1,2)	4 3 2 1	→ (1,2,3)

(c)



(d)

One-Error-Correcting Code

Construction (One-Error-Correcting Rank Modulation Code)

Let $C_1, C_2 \subseteq S_n$ denote two rank modulation codes constructed as follows. Let $A \in S_n$ be a general permutation whose inversion vector is $(x_1, x_2, \dots, x_{n-1})$. Then A is a codeword in C_1 iff the following equation is satisfied:

$$\sum_{i=1}^{n-1} ix_i \equiv 0 \pmod{2n-1}$$

A is a codeword in C_2 iff the following equation is satisfied:

$$\sum_{i=1}^{n-2} ix_i + (n-1) \cdot (-x_{n-1}) \equiv 0 \pmod{2n-1}$$

Between C_1 and C_2 , choose the code with more codewords as the final output.

One-Error-Correcting Code

For the above code, it can be proved that:

- The code can correct one Kendall error.
- The size of the code is at least $\frac{(n-1)!}{2}$.
- The size of the code is at least half of optimal.

Codes Correcting More Errors [1]

Codes correcting more Kendall errors are constructed based on embedding.

First, consider codes of the following form:

- Let $m \geq n - 1$ and let h_1, \dots, h_{n-1} be a set of integers, where $0 < h_i < m$ for $i = 1, \dots, n - 1$. Define the code as follows:

$$\mathcal{C} = \{(x_1, x_2, \dots, x_{n-1}) \mid \sum_{i=1}^{n-1} h_i x_i \equiv 0 \pmod{m}\}$$

[1] A. Barg and A. Mazumdar, "Codes in Permutations and Error Correction for Rank Modulation," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, pp. 854–858, June 2010.

Codes Correcting More Errors

Fact: The above code can correct t Kendall errors if all the syndromes caused by up to t errors are all distinct.

How to find such integers h_1, \dots, h_{n-1} ?

Codes Correcting More Errors

Fact: The above code can correct t Kendall errors if all the syndromes caused by up to t errors are all distinct.

How to find such integers h_1, \dots, h_{n-1} ?

Theorem (Bose-Chowla)

Let q be a power of a prime, and let $m = \frac{q^{t+1}-1}{q-1}$. Then there exist $q+1$ integers $j_0 = 0, j_1, \dots, j_q$ in \mathbb{Z}_m such that the sums

$$j_{i_1} + j_{i_2} + \dots + j_{i_t} \quad (0 \leq i_1 \leq i_2 \leq \dots \leq i_t \leq q)$$

are all different modulo m .

Codes Correcting More Errors

The Bose-Chowla theorem is useful when all the errors in the embedded $(n - 1)$ -dimensional L_1 space are positive errors.

To also handle negative errors, we can “enlarge” the coefficients:

Theorem

For $1 \leq i \leq q + 1$ let

$$h_i = \begin{cases} j_{i-1} + \frac{t-1}{2}m & \text{for odd } t \\ j_{i-1} + \frac{t}{2}m & \text{for even } t \end{cases}$$

where the numbers j_i are given by the Bose-Chowla theorem. Let $m_t = t(t + 1)m$ if t is odd and $m_t = t(t + 2)m$ if t is even. For all $e \in \mathbb{Z}^{q+1}$ such that $\|e\| \leq t$ the sums (i.e., syndromes) $\sum_{i=1}^{q+1} e_i h_i$ are all distinct and nonzero modulo m_t .

Codes Correcting More Errors [1]

More idea: Map each dimension of the $(n - 1)$ -dimensional space to bits using Gray code. Then binary ECC can be turned into ECC for permutations.

[1] A. Mazumdar, A. Barg and G. Zemor, "Constructions of Rank Modulation Codes," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2011.

Capacity of Rank Modulation ECC [1]

Let the number of cells $n \rightarrow \infty$. Consider capacity.

Theorem (Capacity of Rank Modulation ECC)

Let $A(n, d)$ be the maximum number of permutations in S_n with minimum Kendall-tau distance d . We call

$$C(d) = \lim_{n \rightarrow \infty} \frac{\ln A(n, d)}{\ln n!}$$

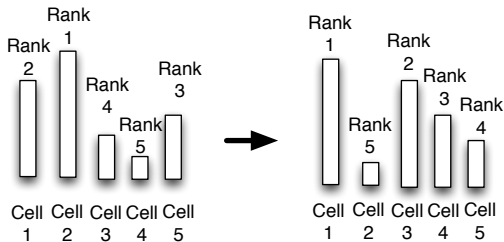
the capacity of rank modulation ECC of Kendall-tau distance d . Then,

$$C(d) = \begin{cases} 1 & \text{if } d = O(n) \\ 1 - \epsilon & \text{if } d = \Theta(n^{1+\epsilon}), 0 < \epsilon < 1 \\ 0 & \text{if } d = \Theta(n^2) \end{cases}$$

Rank Modulation ECC with L_∞ Distance

L_∞ distance between two cell states: The maximum change in the rank of a cell. (We consider the absolute value of the change.)

Example: In the following, cell 2's rank has changed by 4, which is the greatest.

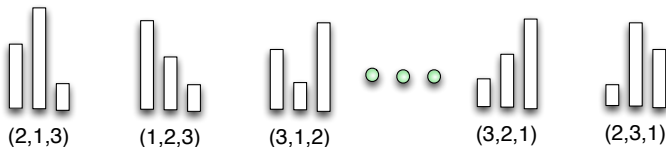


For more, see: I. Tamo and M. Schwartz, "Correcting Limited-Magnitude Errors in the Rank Modulation Scheme," in *IEEE Transactions on Information Theory*, vol. 56, no. 6, pp. 2551-2560, June 2010.

Rank Modulation with Multiple Cell Groups

A practical way to use rank modulation is to partition cells into groups, and apply rank modulation to each cell group. The code, such as ECC, can be defined for all the cell groups together.

Example: Partition cells into groups of size 3.







LDPC code has been studied for this setting, where each cell group is seen as a codeword symbol.





For more, see: F. Zhang, H. Pfister and A. Jiang, "LDPC Codes for Rank Modulation in Flash Memories," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, pp. 859–863, June 2010.

Further reading (1/4)





Error-Correcting Codes for Rank Modulation:

-  A. Jiang, M. Schwartz and J. Bruck, "Error-Correcting Codes for Rank Modulation," *Proc. ISIT*, pp. 1736–1740, 2008.
-  A. Barg and A. Mazumdar, "Codes in Permutations and Error Correction for Rank Modulation," *Proc. ISIT*, pp. 854–858, 2010.
-  A. Jiang, M. Schwartz and J. Bruck, "Correcting Charge-Constrained Errors in the Rank Modulation Scheme," *IEEE Transactions on Information Theory*, vol. 56, no. 5, pp. 2112–2120, 2010.
-  F. Zhang, H. Pfister and A. Jiang, "LDPC Codes for Rank Modulation in Flash Memories," *Proc. ISIT*, pp. 859–863, 2010.

Further reading (2/4)

-  I. Tamo and M. Schwartz, “Correcting Limited-Magnitude Errors in the Rank Modulation Scheme,” *IEEE Transactions on Information Theory*, vol. 56, no. 6, pp. 2551–2560, 2010.
-  A. Barg and A. Mazumdar, “Codes in Permutations and Error-Correction for Rank Modulation,” *IEEE Transactions on Information Theory*, vol. 56, no. 7, pp. 3158–3165, 2010.
-  M. Schwartz and I. Tamo, “Optimal Permutation Anticodes with the Infinity Norm via Permanents of $(0, 1)$ -Matrices,” *Journal of Combinatorial Theory, Series A*, vol. 118, pp. 1761–1774, 2011.
-  A. Mazumdar, A. Barg and G. Zemor, “Constructions of Rank Modulation Codes,” *Proc. ISIT*, 2011.

Further reading (3/4)

-  F. Farnoud, V. Skachek and O. Milenkovic, “Rank Modulation Codes for Translocation Errors,” *Proc. Information Theory and Applications Workshop (ITA)*, San Diego, CA, 2012.
-  M. Schwartz, “Quasi-cross Lattice Tilings with Applications to Flash Memory,” *IEEE Transactions on Information Theory*, vol. 58, no. 4, pp. 2397–2405, 2012.
-  H. Zhou, A. Jiang and J. Bruck, “Systematic Error-correcting Codes for Rank Modulation,” *Proc. ISIT*, 2012.
-  Y. Yehezkeally and M. Schwartz, “Snake-in-the-Box Codes for Rank Modulation,” *IEEE Transactions on Information Theory*, vol. 58, no. 8, pp. 5471–5483, 2012.

Further reading (4/4)



I. Tamo and M. Schwartz, “On the Labeling Problem of Permutation Group Codes under the Infinity Metric,” *IEEE Transactions on Information Theory*, vol. 58, no. 10, pp. 6595–6604, 2012.

Constrained Coding

Issues

- Delay onset of errors
- Improve reliability
- Improve write access
- Increase storage capacity
- Reduce interference

Techniques

- Error correction codes
- Codes for rewriting data
- Rank modulation (with rewriting)
- **Constrained coding**

Issues

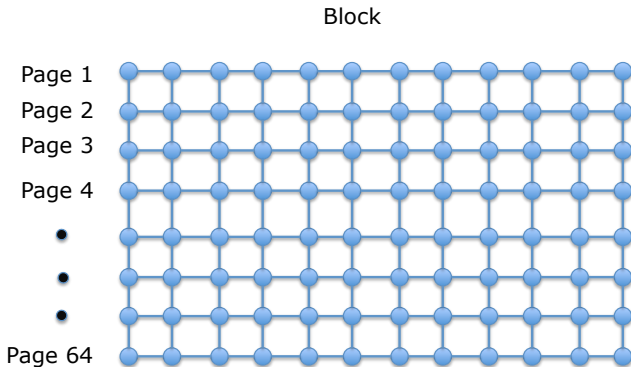
- Delay onset of errors
- **Improve reliability**
- Improve write access
- Increase storage capacity
- **Reduce interference**

Techniques

- Error correction codes
- Codes for rewriting data
- Rank modulation (with rewriting)
- **Constrained coding**

Example of NAND Flash Memory Block

Example of a NAND flash memory block: A block has ~ 64 pages.
A page has thousands of cells.



Intercell Coupling in NAND Flash Memories

- Intercell Coupling: The threshold voltage V_{th} is shifted due to parasitic capacitance between neighboring cells, including:
 - Horizontal direction (bit-line to bit-line coupling)
 - Vertical direction (word-line to word-line coupling)
 - Diagonal direction

Intercell Coupling in NAND Flash Memories

- Intercell Coupling: The threshold voltage V_{th} is shifted due to parasitic capacitance between neighboring cells, including:
 - Horizontal direction (bit-line to bit-line coupling)
 - Vertical direction (word-line to word-line coupling)
 - Diagonal direction
- The amount of shift in V_{th} is affected by:
 - Coupling coefficient C
 - V_{th} shift of neighboring cell due to programming

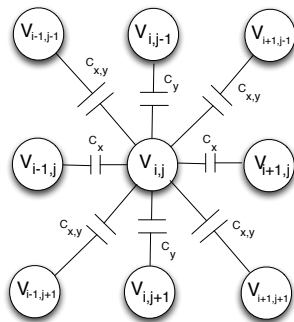
Intercell Coupling in NAND Flash Memories

- Intercell Coupling: The threshold voltage V_{th} is shifted due to parasitic capacitance between neighboring cells, including:
 - Horizontal direction (bit-line to bit-line coupling)
 - Vertical direction (word-line to word-line coupling)
 - Diagonal direction
- The amount of shift in V_{th} is affected by:
 - Coupling coefficient C
 - V_{th} shift of neighboring cell due to programming
- When memory density scales up, intercell coupling becomes more severe.

Intercell Coupling in NAND Flash Memories

The V_{th} shift of middle cell caused by shifting of neighboring cells is

$$\Delta V_{i,j} = C_x(\Delta V_{i-1,j} + \Delta V_{i+1,j}) + C_y(\Delta V_{i,j-1} + \Delta V_{i,j+1}) + C_{x,y}(\Delta V_{i-1,j-1} + \Delta V_{i+1,j-1} + \Delta V_{i-1,j+1} + \Delta V_{i+1,j+1})$$



Constrained Coding Techniques

Consider MLC with q levels: $0, 1, \dots, q - 1$.

- Proposed Constraint [1]: A cell of level 0 cannot be adjacent to a cell of level $q - 1$.

Example: When $q = 4$, two cells of level 0 and level 3 cannot be adjacent.

Constrained Coding Techniques

Consider MLC with q levels: $0, 1, \dots, q - 1$.

- Proposed Constraint [1]: A cell of level 0 cannot be adjacent to a cell of level $q - 1$.





Example: When $q = 4$, two cells of level 0 and level 3 cannot be adjacent.

- Proposed Constraint [2]: For every cell, the difference between its own level and the summation of its neighboring cells' levels cannot be too large.



[1] Y. Kim, K. Son, K. L. Cho, J. Kim, J. J. Kong, and J. Lee, "RLL Codes for Flash Memory," presented in JCCI, 2010.

[2] A. Berman and Y. Birk, "Mitigating inter-cell coupling effects in MLC NAND flash via constrained coding," presented at Flash Memory Summit, 2010.

Further reading (1/2)

-  A. Berman and Y. Birk, “Mitigating Inter-cell Coupling Effects in MLC NAND Flash via Constrained Coding,” presented at Flash Memory Summit, August 2010.
-  Y. Kim, K. Son, K. L. Cho, J. Kim, J. J. Kong, and J. Lee, “RLL Codes for Flash Memory,” presented in JCCI, 2010.
-  A. Jiang, J. Bruck and H. Li, “Constrained Codes for Phase-change Memories,” *Proc. ITW*, 2010.
-  A. Berman and Y. Birk, “Error Correction Scheme for Constrained Inter-Cell Coupling in Flash Memory,” presented at Non-Volatile Memories Workshop (NVMW), San Diego, CA, March 2011.

Further reading (2/2)

-  M. Qin, E. Yaakobi and P. Siegel, “Time-space Constrained Codes for Phase Change Memories,” presented at Flash Memory Summit, 2011.
-  R. Motwani, “Hierarchical Constrained Coding for Floating-gate to Floating-gate coupling Mitigation in Flash Memory,” Globecom 2011.

Summary and Future Directions

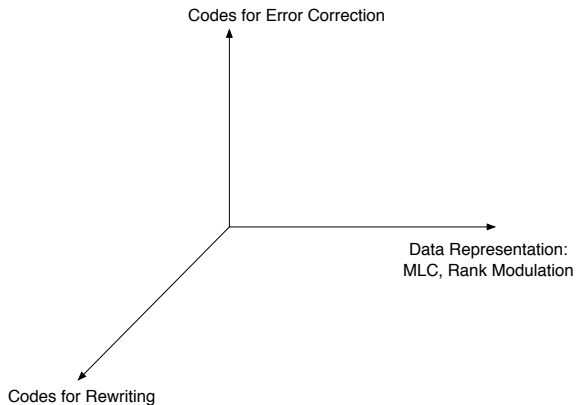
Issues

- Delay onset of errors
- Improve reliability
- Improve write access
- Increase storage capacity
- Reduce interference

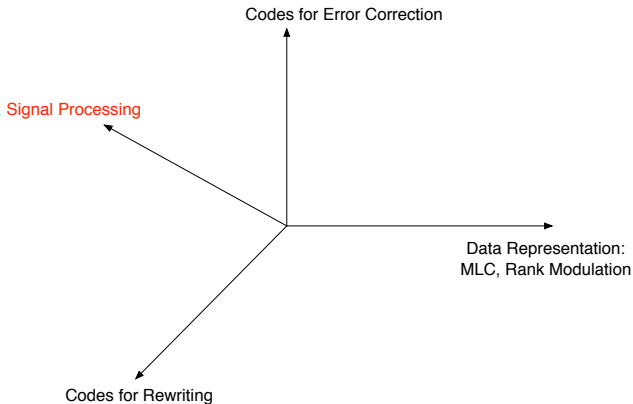
Techniques

- Error correction codes
- Codes for rewriting data
- Rank modulation
- Constrained coding

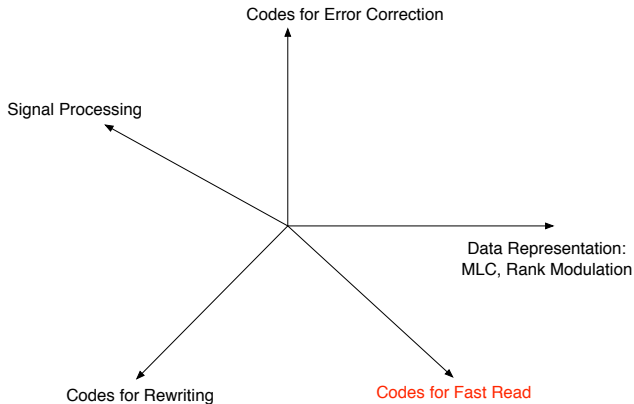
Open Problems on Coding for NVMs



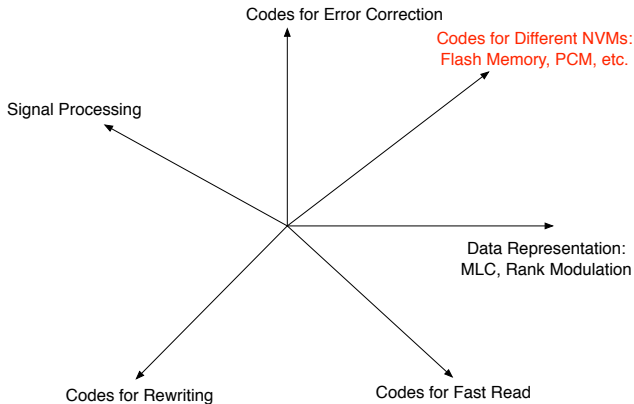
Open Problems on Coding for NVMs



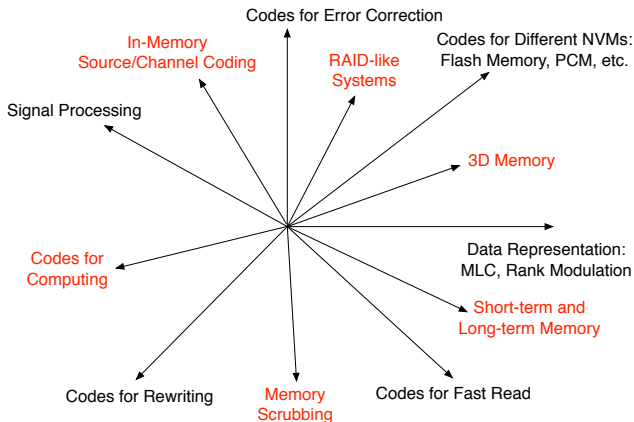
Open Problems on Coding for NVMs



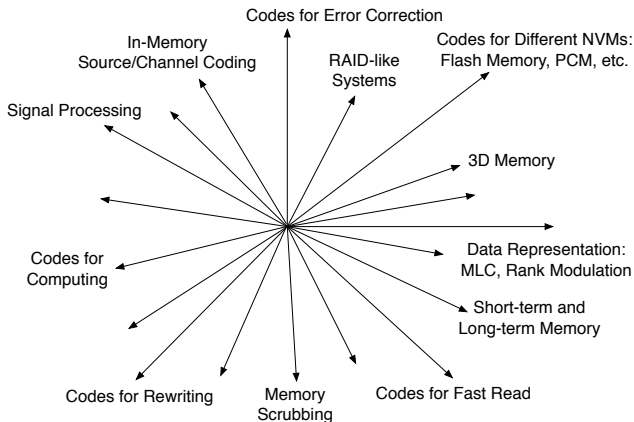
Open Problems on Coding for NVMs



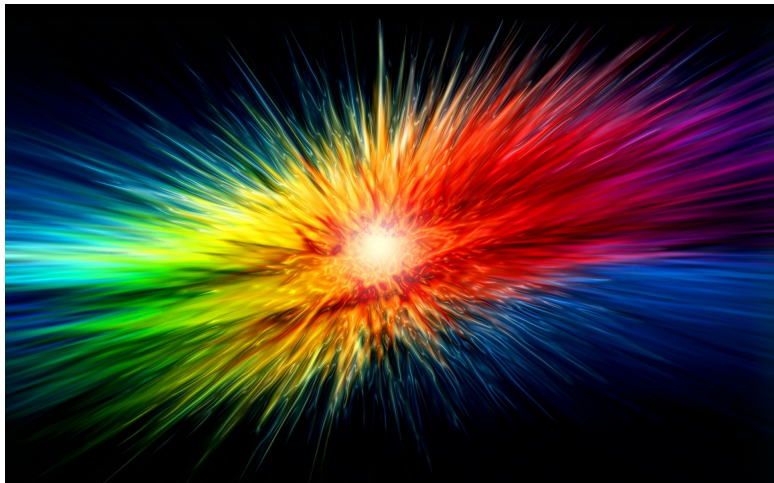
Open Problems on Coding for NVMs



Open Problems on Coding for NVMs



Open Problems on Coding for NVMs



Research Problems

- 1 Design of graph-based non-binary codes for Flash channels with respect to spatio-temporal variability.
- 2 Design of LDPC codes in the high-reliability region of quantized Flash channels with respect to absorbing/trapping sets.
- 3 Error Correction coding methods and information theory for STTRAM.
- 4 Design of codes combining WOM, ECC and ICI properties.
- 5 Probabilistic vs. worst case analysis of various coding methods for NVM-inspired channels.

IEEE JSAC Special Issue on Communication Methodologies for the Next-Generation Storage Systems

Topics include:

- Device-level channel modeling for emerging storage technologies
- Information theory and fundamental data transmission limits for new storage channels
- Practical coding and signal processing methods cognizant of underlying physical constraints
- Architecture and design of large-scale storage subsystems based on new non-volatile memories
- Security, data compression and communication techniques for cloud storage and distributed storage networks

Deadline: May 1st, 2013

2013 Non-Volatile Memories Workshop (NVMW)

March 3-5, 2013, University of California, San Diego

The workshop solicits presentations on any topic related to non-volatile, solid state memories, including:

- Advances in memory devices or memory cell design.
- Characterization of commercial or experimental memory devices.
- Error correction and data encoding schemes for non-volatile memories.
- Advances in non-volatile memory-based storage system.
- Operating system and file system designs for non-volatile memories.
- Security and reliability of solid-state storage systems.
- Applications of non-volatile memories to scientific, “big data,” and high-performance workloads.
- Implications of non-volatile memories for applications such as databases and NoSQL systems.

Deadline: November 19, 2012

NVMW 2013

4th Non-Volatile Memories Workshop



March 3-5, 2013

UC San Diego

La Jolla, California USA

<http://nvmw.ucsd.edu>



NVSL
Non-volatile Systems Laboratory

